Clustering aplicado al problema VRP con múltiples depósitos y ventanas de tiempo.

Silvia Fojo ***Viviana García***

Tutores: Ing. Libertad Tansini
Ing. Omar Viera

Informe Final, Taller V 2000
Departamento de Investigación Operativa
Instituto de Computación
Facultad de Ingeniería

INDICE

I. INTRODUCCIÓN **5**
II. ANALISIS DE REQUERIMIENTOS **8**
1. OBJETIVO **8**
2. DEFINICIÓN DEL PROBLEMA **8**
3. ANÁLISIS **9**
3.1 CRITERIOS DE DISTANCIA ENTRE PUNTOS 9
3.2 ALGORITMOS 10
3.2.1 Aglomeración 10
3.2.2 Algoritmo Rock 12
3.3 INGRESO DE DATOS DE ENTRADA Y VISUALIZACIÓN DE LOS RESULTADOS 13
3.4 FORMATO ENTRADA/SALIDA DE LOS ALGORITMOS 13
3.5 EVALUACIÓN DE RESULTADOS 14
3.5.1 Algoritmo de ruteo 14
III. ANÁLISIS DE FUNCIONALIDAD **16**
1. OBJETIVO **16**
2. ANÁLISIS **16**
2.1 INGRESO DE LOS DATOS 16
2.2 EJECUCIÓN DE LOS ALGORITMOS DE CLUSTERING 16
2.3 SALIDA Y VISUALIZACIÓN GRÁFICA DE LOS RESULTADOS DE CLUSTERING 17
2.4 EJECUCIÓN DE HEURÍSTICAS DE ASIGNACIÓN 17
2.5 VISUALIZACIÓN DEL COSTO DE RUTEO 18
IV. ADAPTACIÓN AL PROBLEMA **19**
1. OBJETIVO **19**
2. ANÁLISIS **19**
2.1 CONSIDERACIONES PARA APLICAR ALGORITMOS AGLOMERATIVOS 19
2.1.1 Conjunto de clusters de partida 19
2.1.2 Combinación de clusters 20
2.1.3 Criterio de parada 20
2.2 CONSIDERACIONES PARA APLICAR EL ALGORITMO ROCK 21
2.2.1 La función objetivo 21
2.2.2 Estimación de f(θ) 22
2.3 CONCLUSIONES 22
V. DISEÑO **24**
1. OBJETIVO **24**
2. SEUDOCÓDIGO **24**
2.1 ALGORITMO ROCK 24
2.2 ALGORITMO AGLOMERACION 25
3. DISEÑO DE MÓDULOS **27**
3.1 Definición de Tipos 27
3.2 Definición de Módulos 27
VI. IMPLEMENTACIÓN **33**
1. OBJETIVO **33**
2. ANÁLISIS **33**
2.1 LENGUAJES DE PROGRAMACIÓN 33
3. ESTRUCTURAS DE DATOS **33**
4. GENERALIDADES **33**
VII VERIFICACIÓN **35**
1. OBJETIVO **35**
2. TESTEO DE MÓDULOS **35**
VIII. EVALUACIÓN DE LOS RESULTADOS **37**
1. OBJETIVOS **37**
2. CRITERIOS DE BONDAD **37**
2.1 DISTANCIA INTRACLUSTER 38
2.2 DISTANCIA INTERCLUSTER 39
2.3 LARGO DE RUTA 39
3. RESULTADOS **39**
4. CÓMO VARIA LA FORMA DE UN CLUSTER **40**

4.1 CASO 1 42
4.2 CASO 2 44
4.3 CASO 7 45
4.4 CASO 10 47
5 CONCLUSIONES DE LOS RESULTADOS OBTENIDOS 50
IX. TRABAJO FUTURO 53
RESUMEN DEL PROYECTO 54
BIBLIOGRAFÍA 55
ANEXO I - ESTADO DEL ARTE DEL CLUSTERING 57
ANEXO II - SOFTWARE DE CLUSTERING. 58
ANEXO III – CASOS COMPLEMENTARIOS 59
ANEXO IV – MANUAL DE USUARIO 60
ANEXO V – DIAGRAMA DE GANTT 61

Resumen

El Problema de Ruteo de Vehículos con Múltiples Depósitos y Ventanas de Tiempo (MDVRPTW por sus siglas en inglés) consiste básicamente en encontrar las rutas óptimas entre clientes y depósitos, teniendo en cuenta horarios de servicio, producción de cada cliente y capacidad máxima de cada depósito.

Una forma de resolver el MDVRPTW consiste en dividir el problema en dos fases: la primera es asignar un subconjunto de los clientes a cada depósito y la segunda rutear, por separado, en cada uno de estos subconjuntos obtenidos. El objetivo de este estudio es enfocar la atención en la primera de estas fases, la fase de asignación, considerando la aplicación de una de las técnicas más conocidas de clustering, llamada clustering jerárquico. Para ello se han seleccionado dos algoritmos, uno de éstos consiste en un algoritmo clásico de clustering jerárquico llamado: AGLOMERACIÓN y el otro es una adaptación del algoritmo ROCK.

Para evaluar la aplicabilidad de estos algoritmos de clustering al MDVRPTW, se han comparado los resultados de un ruteo utilizando como algoritmos de asignación Aglomeración y Rock contra los resultados de un ruteo utilizando heurísticas de asignación. Tales heurísticas fueron desarrolladas en el Departamento de Investigación Operativa por Libertad Tansini, Daniel Giosa y Omar Viera específicamente para resolver el subproblema de la asignación. Los resultados obtenidos con estas comparaciones muestran que en el 87% de los casos se obtuvo menor costo total de ruteo usando clustering para la fase de asignación, aunque para los casos de prueba de gran tamaño los tiempos de ejecución favorecen a las heurísticas. Por lo tanto, para problemas de tamaño mediano y donde el tiempo de ejecución no es un factor crítico, es factible utilizar clustering como algoritmo de asignación previo al ruteo.

I. INTRODUCCIÓN

Este trabajo se realiza en el marco de un proyecto de Taller V de la carrera de Ingeniería en Computación de la Facultad de Ingeniería. Este proyecto surge como propuesta del Departamento de Investigación Operativa del Instituto de Computación a raíz de la tesis de Maestría de Libertad Tansini. Esta tesis estudia algoritmos de asignación para el MDVRPTW.

El problema de MDVRPTW consiste en determinar un conjunto de rutas de forma tal que:
- cada ruta comienza y termina en un mismo depósito;
- los requerimientos de los clientes son factibles de ser atendidos en una sola visita de un vehículo;
- el horario de servicio, o ventana de tiempo, de clientes y depósitos son respetadas;
- la suma de los requerimientos de los clientes que son visitados por un mismo vehículo no supera la capacidad del mismo;
- la suma de los requerimientos de los clientes asignados a un mismo depósito no supera la capacidad, o demanda, del mismo;
- el costo total es minimizado.

Una forma de resolver este problema es dividiéndolo en dos fases: Asignación y Ruteo. La primera fase, consiste en asignar a cada depósito un subconjunto de los clientes, y la segunda en construir las rutas más convenientes que unen los depósitos con sus clientes asignados [13].

El problema particular que nos ocupa considera las características de MDVRPTW, pero se incorpora una restricción más: la compatibilidad entre clientes y depósitos. Un ejemplo de este problema es el planteado al Departamento de Investigación Operativa hace algunos años por Conaprole. El problema de recolección de leche a granel consiste en un conjunto de tambos, de los cuales se debe transportar su producción diaria de leche hacia las plantas procesadoras. Tanto tambos como plantas procesadoras atienden solamente dentro de un determinado horario de servicio. Cada tambo tiene una cantidad limitada de producción de leche, y a su vez cada planta procesadora tiene una capacidad máxima que no debe ser superada. Cada planta procesadora se especializa en productos de cierto tipo, lo que impone ciertas condiciones de calidad sobre la leche para que sea posible su utilización en la planta. Por ejemplo, una planta procesadora puede estar dedicada a la producción de quesos finos, para los cuales es indispensable utilizar leche de un nivel de calidad que solo es producida en algunos tambos. En este caso, se dice que los tambos cuya leche es de la calidad requerida son “compatibles” con la planta procesadora dedicada a quesos finos. En caso contrario, se dice que el tambo es incompatible con la planta procesadora. Por esta razón, solo aquellos tambos que son compatibles con la planta procesadora pueden ser asignados a ella.

De acuerdo a las características presentadas, estamos ante un problema de recolección, puesto que se debe recolectar la producción de los clientes y transportarla hacia los depósitos. En este caso los clientes son los tambos y los depósitos son las plantas procesadoras.
Como se dijo antes, el problema puede dividirse en dos fases bien diferenciadas. La primera es la fase de asignación y consiste en hallar una asignación conveniente de clientes a depósitos. La segunda, es la fase de ruteo y consiste en construir las rutas de forma tal que una todos los clientes asignados a un mismo depósito.

Este trabajo se centra en la resolución de la primera de las fases, la Fase de Asignación. Actualmente, existen pocos algoritmos estudiados para resolver el problema de asignación, básicamente se cuenta con algunas técnicas típicas [14] más otras desarrolladas en el Departamento de Investigación Operativa por Libertad Tansini, Daniel Giosa y Omar Viera [13].
El objetivo de este trabajo es continuar el estudio de algoritmos de asignación, en particular estudiando la aplicación de técnicas de clustering a este problema. Específicamente se estudia la aplicación de algoritmos de clustering jerárquico.

Clustering es un área ampliamente estudiada y se han propuesto varias definiciones de cluster para adecuarse a diferentes aplicaciones (ver Anexo I).
En general, el objetivo de clustering es encontrar grupos interesantes (clusters) en el conjunto de datos de manera que puntos pertenecientes al mismo grupo son mas cercanos entre sí que entre puntos de otros grupos. En nuestro caso, los puntos son los clientes más los depósitos y se han agregado algunas restricciones adicionales a este problema. Por ejemplo, que cada cluster contenga

a lo sumo un depósito, que se respete la compatibilidad de clientes y depósitos, que la demanda de un depósito no sea superada por el total de la producción de los clientes asignados a su cluster. Además de tener en cuenta el horario de servicio (ventana de tiempo) de cada cliente y cada depósito.

Se ha realizado una búsqueda y un estudio de las distintas técnicas y algoritmos de clustering existentes hoy en día. Si bien son técnicas conocidas desde hace mucho tiempo, principalmente en aplicaciones de estadísticas multivariadas, estás técnicas han despertado últimamente un interés mayor con el advenimiento de la tecnología de Data Mining. Por lo tanto, muchas de éstas técnicas fueron específicamente desarrolladas pensando en ésta área.
Existen principalmente dos enfoques para algoritmos de clustering. Están aquellos basados en algoritmos del tipo K-Means [1] y los basados en enfoques jerárquicos [1]. Para este taller hemos elegido concentrarnos en el estudio de algoritmos de clustering de tipo jerárquico (otros talleres se han dedicado al estudio del enfoque K-Means). Estos algoritmos trabajan por aglomeración. Se comienza con un cluster para cada uno de los puntos, es decir, cada punto forma su propio cluster y luego gradualmente se combinan hasta que todos los puntos se hayan acumulado en un solo cluster.
Hacia el comienzo del proceso, los clusters tienen pocos elementos pero muy cercanos entre ellos. Hacia el final del proceso, los clusters son grandes y sus elementos pueden estar bastante alejados entre si. Necesariamente se han realizado adaptaciones de estos algoritmos para adecuarlos al problema particular de asignación para el Ruteo de Vehículos.

Para evaluar los resultados obtenidos, se han realizado pruebas en términos de criterios de bondad de clusters y también en términos del costo total del ruteo luego de hallar diferentes asignaciones.
Los criterios de bondad utilizados, son la distancia intracluster e intercluster.
El algoritmo de ruteo utilizado fue proporcionado por el usuario: Libertad Tansini y es una adaptación del algoritmo Clark & Wright [14].

Se han comparado los resultados del ruteo luego de haber hallado una asignación utilizando AGLOMERACION, ROCK y las técnicas de asignación desarrolladas en el departamento de Investigación Operativa [13]. Los resultados obtenidos con estas comparaciones, muestran que en el 87% de los casos se obtuvo menor costo total de ruteo usando clustering para la fase de asignación, aunque para casos de gran tamaño los tiempos de ejecución favorecen a las heurísticas. El orden de tiempo de ejecución en el peor caso para AGLOMERACION es de $O(n^2 + n^2 \log n)$ y para ROCK es $O(n^2 + n(n-1)m_a + n^2 \log n)$ donde m_a es la cantidad promedio de vecinos y n es la cantidad total de puntos (clientes y depósitos). Los ordenes de ejecución de las heurísticas varían según cuál de ellas se utilice pero en todos los casos son sensiblemente menores [13]. Podemos considerar entonces, que para problemas de tamaño mediano y donde el tiempo de ejecución no sea un factor crítico, es factible utilizar clustering como algoritmo de asignación previo al ruteo.

Resulta de gran interés entonces, investigar algoritmos de clustering para aplicarlos al problema MDVRPTW. En cuanto a la inclusión de la ventana de tiempo en los algoritmos de clustering se probaron varias formas de incluir este dato en el cálculo de distancia entre puntos. El algoritmo Aglomeración [1] y la distancia Ángulo [1], utilizada como distancia entre puntos fue la combinación de algoritmo y función de distancia con la que se obtuvo los mejores resultados. Tanto en lo que tiene que ver con criterios de bondad típicos de clustering como en cuanto a los resultados posteriores en el ruteo.

Este informe esta organizado de la siguiente forma: en la Sección II se presenta el análisis de los requerimientos que surgieron a partir de las reuniones con el usuario, en este caso, Libertad Tansini. En la Sección III se presenta el análisis de funcionalidad que detalla las funciones necesarias para cumplir con los requerimientos del usuario. La Sección IV Adaptación al problema presenta el análisis que surge de la necesidad de adaptar los algoritmos de clustering seleccionados para su aplicación al problema. Se discuten distintas alternativas y las decisiones finalmente tomadas para la implementación de los algoritmos. La Sección V presenta el diseño de módulos, las estructuras auxiliares que se utilizaron y el seudocódigo de los algoritmos a implementar. La Sección VI esta enfocada a los aspectos de implementación de los algoritmos, la selección de la plataforma y lenguajes de programación.. La sección VII presenta un resumen sobre como se realizó la verificación de los diferentes módulos identificados en el diseño.

En la Sección VIII se presentan los resultados y estudios comparativos obtenidos a partir de la aplicación de los distintos algoritmos de asignación en casos de distinto orden de magnitud. Finalmente la sección IX presenta posibles extensiones a este trabajo.
También se incluyen anexos que presentan el Estado del Arte del Clustering (Anexo I), una investigación sobre software que realiza clustering (Anexo II), mas casos de pruebas y los resultados obtenidos (Anexo III), el manual de usuario de la aplicación implementada para ingresar los datos, ejecutar los algoritmos y visualizar resultados (Anexo IV) y por último un diagrama de Gantt con un cronograma de este taller.

II. ANALISIS DE REQUERIMIENTOS

1. Objetivo

El objetivo de esta sección es definir los requerimientos para cumplir con las necesidades del usuario.

Este análisis de requerimientos es el resultado de las reuniones mantenidas con el Tutor: Omar Viera y el usuario Libertad Tansini. La propuesta de este trabajo surge a raíz de la tesis de Maestría de Libertad Tansini.

Se requiere una herramienta que permita aplicar algoritmos de clustering, para realizar un pretratamiento a ciertos datos que originalmente serían la entrada a algoritmos específicos de ruteo con múltiples depósitos y ventanas de tiempo (MDVRPTW). De esta forma se busca agrupar en clusters aquellos puntos que poseen ciertas características, que conduzcan a la obtención de mejores resultados en la aplicación de los algoritmos de ruteo. Por ejemplo, en la figura 1 y 2 se presentan dos posibles resultados de agrupación y ruteo sobre un conjunto de clientes y dos depósitos (los círculos mas grandes corresponden a depósitos y los más pequeños a clientes). Obsérvese que a partir de la agrupación de la figura 2 se obtuvo menor costo total de ruteo que con la agrupación de la figura 1. Resulta de interés, entonces, poder determinar a partir de la herramienta, el mejor clustering o agrupación en términos de costo total de ruteo.

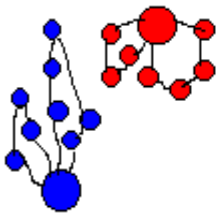


Figura 1: Costo total de ruteo = 120

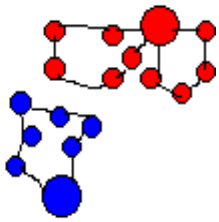


Figura 2: Costo total de ruteo = 100

En las secciones siguientes, se definen los requerimientos que debe cumplir la herramienta a desarrollar para cubrir las necesidades del usuario.

2. Definición del problema

El problema de MDVRPTW [13] consiste en determinar un conjunto de rutas de vehículos de forma tal que:

- cada ruta comienza y termina en un mismo depósito;
- los requerimientos de los clientes son factibles de ser atendidos en una sola visita de un vehículo;
- el horario de servicio, o ventana de tiempo, de clientes y depósitos son respetadas;
- la suma de los requerimientos de los clientes que son visitados por un mismo vehículo no supera la capacidad del mismo;
- la suma de los requerimientos de los clientes asignados a un mismo depósito no supera la capacidad, o demanda, del mismo;
- el costo total es minimizado.

En el MDVRPTW, los clientes deben ser asignados a un solo depósito. Cada vehículo debe partir y volver al mismo depósito. Para cada depósito, la cantidad de vehículos de la flota varía entre un mínimo y un máximo específicos. El problema MDVRPTW es NP-Duro [13] y resulta de mucho interés el desarrollo de algoritmos heurísticos para esta clase de problema.

MDVRPTW puede verse como un problema de clustering en el sentido de que la salida es un conjunto de vehículos agrupados por depósitos. Esta interpretación sugiere que una forma de resolverlo usando clustering es agrupar (clusterizar) los clientes y luego hallar las rutas en cada cluster. Una forma de resolver éste problema es dividiéndolo en dos partes: Asignación y Ruteo. La primera parte,

consiste en asignar a cada depósito un subconjunto de los clientes, y la segunda en construir las rutas, más convenientes, que unen los depósitos a sus clientes asignados.
Este proyecto trata la fase de Asignación, buscando la forma de aplicar técnicas de clustering para resolverla.

3. Análisis

Se requiere una herramienta que permita visualizar y comparar gráficamente los resultados de la aplicación de diferentes algoritmos de clustering a un conjunto de puntos que originalmente serían las entrada a un algoritmo de ruteo. Estos algoritmos procesarán este conjunto de datos, y guardarán los resultados en archivos de texto con un formato específico. Esto permitirá su utilización como entrada para algoritmos particulares de ruteo. El formato de tales archivos será definido más adelante en este documento.

Los datos de entrada consisten en un conjunto de clientes y un conjunto de depósitos de los cuáles se conoce su posición con respecto a un origen de coordenadas. De los clientes se conoce además el horario de atención, su producción y una lista de los depósitos que no reciben su producción (incompatibles). Mientras que de los depósitos, se conoce su demanda máxima y su horario de atención.

Se buscan clusters de clientes y depósitos de forma tal de minimizar distancias de acuerdo a la posición y horario de atención. Debiéndose cumplir, además, que cada cluster contiene a lo sumo un depósito. También se tendrá en cuenta, que la demanda de cada uno de los depósitos no sea superada por el total de la producción de los clientes asignados y respetando la compatibilidad cliente - depósito.

3.1 Criterios de distancia entre puntos

Uno de los puntos clave de este trabajo es encontrar la forma de incluir el horario de atención o ventana de tiempo al momento de hallar los clusters, ya que debe tenerse en cuenta al momento de decidir si dos puntos con diferentes ventanas de tiempo deben pertenecer a un mismo cluster. Para ello, este dato se va a tomar como el centro de la ventana de tiempo en minutos, contados desde la hora cero. La forma tratar este dato en los algoritmos de clustering es incluirlo en la función de distancia entre puntos.

Como se muestra en el Estado del Arte del Clustering (ver Anexo I), se puede definir un gran número de funciones para el cálculo de la distancia entre dos puntos. Para la evaluación de los resultados de clustering, se propone estudiar las funciones de distancia que se listan a continuación y a su vez comparar las diferencias en los clusters resultado (en términos de criterios de bondad de clusters y de costo de ruteo) cuando se utilizan diferentes medidas de distancia entre puntos.

1) Suma ponderada

Dados dos puntos en el plano, $X = (x_1, x_2)$ e $Y = (y_1, y_2)$ y t_x , t_y el centro de sus ventanas de tiempo.

$$dist(X,Y) = w_{xy}(\Sigma i\ (x_i - y_i)^{2})^{1/2} + w_t \left| t_x - t_y \right|$$

donde w_{xy} y w_t son las ponderaciones y serán ingresadas por el usuario.

El uso de esta distancia euclídea tiene la ventaja que la distancia entre dos objetos cualesquiera no se ve afectada cuando se agregan nuevos objetos, que pueden ser outliers, al análisis.

2) Distancia de cuadra (City-block , Manhattan)

Dados dos puntos $X = (x_1, x_2, x_3)$ e $Y = (y_1, y_2, y_3)$

$$dist(X,Y) = \Sigma i \left| x_i - y_i \right|$$

donde en nuestro caso la tercer coordenada representa el centro de la ventana de tiempo.

Es igual que la distancia euclídea solo que no se pondera el efecto de las grandes diferencias.

3) Distancia de Chebychev

Dados dos puntos $X = (x_1, x_2, x_3)$ e $Y = (y_1, y_2, y_3)$

dist (X,Y) = Máx. $|xi - yi|$, $1 \leq i \leq 3$

donde en nuestro caso la tercer coordenada representa el centro de la ventana de tiempo.

Se utiliza cuando se define que dos objetos son diferentes si son diferentes en cualquiera de sus dimensiones.

4) Angulo entre dos vectores

Dados dos puntos $X = (x_1, y_1, t_1)$ e $Y = (x_2, y_2, t_2)$

sim (X, Y) = sen(w)

donde w es el ángulo formado por los vectores correspondientes a los puntos X e Y.

El ángulo w se halla a partir de la sig. fórmula

$W = \text{Cos}^{-1}(A*A' + B*B' + C*C')$

Donde $A = x1/ r$, $B = y1 / r$, $C = t_1 / r$

$A' = x2/ r'$, $B' = y2 / r'$, $C' = t_2 / r'$

Donde $r = (x1^2 + y1^2 + t_1 ^2)^{½}$

$r' = (x2^2 + y2^2 + t_2 ^2)^{½}$

En este caso se considera dos registros como altamente asociados debido a la similitud en la forma en que los campos dentro de cada registro están relacionados.

La solución es utilizar una interpretación geométrica diferente de los mismos datos. En lugar de pensar en x e y como puntos en el espacio y medir la distancia entre ellos, pensamos en ellos como vectores y medimos el ángulo que forman.

3.2 Algoritmos

En general, los algoritmos de clustering buscan la maximización de una cierta función objetivo que relaciona la posición de los clientes y depósitos, teniendo en cuenta el centro de una ventana de tiempo, la cual representa el horario de atención de cada uno de ellos. Esta función objetivo dependerá del algoritmo a aplicar y de los diferentes criterios de distancia entre puntos y entre clusters que se utilicen.

Existen principalmente dos enfoques para algoritmos de clustering. Están aquellos basados en algoritmos del tipo K-Means y los basados en enfoques jerárquicos. Para este taller hemos elegido concentrarnos en el estudio de algoritmos de clustering de tipo jerárquico (otros talleres se han dedicado al estudio del enfoque K-Means). Son algoritmos que trabajan por aglomeración. Se comienza con un cluster para cada uno de los puntos, es decir, cada punto forma su propio cluster y luego gradualmente se combinan hasta que todos los puntos se hayan acumulado en un solo cluster. Hacia el comienzo del proceso los clusters son muy pequeños y tienen pocos elementos pero muy cercanos entre ellos. Hacia el final del proceso, los clusters son grandes y cada vez menos definidos. Necesariamente se han realizado adaptaciones de estos algoritmos para adecuarlos al problema particular de asignación para el ruteo de vehículos. Estas adaptaciones se especifican en la Sección IV.

En particular, el usuario de este proyecto ha solicitado que la herramienta a desarrollar permita la aplicación de dos algoritmos de clustering jerárquico: AGLOMERACIÓN y ROCK.

3.2.1 Aglomeración

El algoritmo de AGLOMERACIÓN consiste en una técnica que comienza, en general, con un cluster por cada punto, y en cada paso combina aquellos clusters que optimizan una función objetivo. Se pueden obtener variantes de este algoritmo definiendo distintas funciones objetivo. Esto se logra tomando distintos criterios de distancia entre clusters, pues este algoritmo busca combinar aquellos clusters cuya distancia es mínima.

3.2.1.1 Criterios de distancia entre clusters

Existen muchos criterios para el cálculo de la distancia entre clusters. La elección de diferentes medidas de distancia determina distintos resultados de clustering. Algunos de los criterios de distancia entre clusters conocidos son los siguientes:

Single linkage
La distancia entre dos clusters está dada por la mínima distancia entre dos puntos de clusters distintos. Tal como se muestra en la figura 3, la distancia entre el cluster rojo y el azul sería la distancia entre los puntos señalados por la flecha. Este método produce clusters con la propiedad de que todo miembro de un cluster se encuentra más cerca de algún miembro de su mismo cluster que de cualquier otro punto fuera de él.

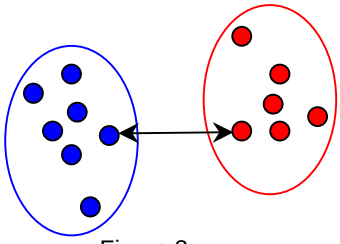


Figura 3

Complete linkage
La distancia entre dos clusters está dada por la máxima distancia entre dos puntos de clusters distintos. Tal como se muestra en la figura 4, la distancia entre el cluster rojo y el azul sería la distancia entre los puntos señalados por la flecha. Este método y el anterior encuentra clusters aunque estos sean de distintos tamaños y formas. Sin embargo, son altamente sensibles a los outliers y el ruido.

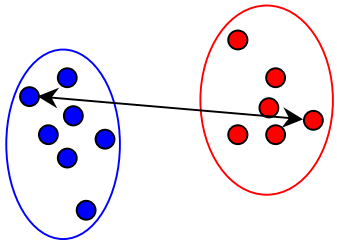


Figura 4

Comparación de centros no ponderados (UPGMC)
La distancia entre dos clusters está dada por la distancia entre sus centros (elemento promedio). Tal como se muestra en la figura 5, la distancia entre el cluster rojo y el azul sería la distancia entre las cruces señaladas por la flecha. Estas cruces representan el elemento promedio del cluster. Este método tiende a fallar cuando los clusters tienen formas arbitrarias y muy diferentes tamaños.

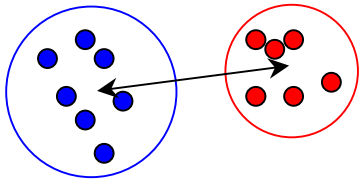


Figura 5

Promedio de grupos de pares no ponderados (UPGMA)
La distancia entre dos clusters es el promedio de distancia entre todo par de puntos de clusters distintos.

Promedio de grupos de pares ponderados (WPGMA)
Igual al anterior excepto que se introduce una ponderación para tomar en cuenta las diferencias en los tamaños de los clusters. Es mejor que el anterior cuando se sabe que el tamaño de los clusters es muy irregular.

Comparación de centros ponderados (WPGMC)
Igual a UPGMC excepto que se introduce una ponderación para tomar en cuenta las diferencias en los tamaños de los clusters. Este método es mejor que UPGMC cuando se sabe que el tamaño de los clusters es muy irregular.

Within Groups
Se busca aquel par de clusters que al combinarlos produce un cluster de varianza mínima. Es similar a UPGMA excepto que los clusters que se fusionan, se eligen de forma tal que la varianza dentro del cluster sea minimizada. Esto tiende a producir clusters mas “compactos” que con UPGMA.

Método Ward
Igual al anterior excepto que busca minimizar la suma de los cuadrados de la varianza. Usa el análisis de varianza para evaluar las distancias entre clusters. Intenta minimizar la suma de los cuadrados (Sum of Squares - SS) de cualquier par de clusters que pueden formarse en cada paso. Este método es muy eficiente aunque tiende a crear clusters muy pequeños.

Para el caso de AGLOMERACIÓN, las combinaciones de criterios de distancia de puntos y clusters que debería ofrecer la herramienta son las siguientes:

	Suma Ponderada	Distancia de cuadra	Distancia de Chebychev	Angulo
Single linkage	SI	SI	SI	SI
Complete linkage	SI	SI	SI	SI
UPGMC	SI	SI	SI	SI
WPGMC	SI	SI	SI	NO
UPGMA	SI	SI	SI	NO
WPGMA	SI	SI	SI	NO
Within Groups	SI	SI	SI	NO
Ward	SI	SI	SI	NO

3.2.2 Algoritmo Rock

Como se detalla en el Estado del Arte de Clustering (Anexo I), Rock es un algoritmo de clustering aglomerativo, que se basa en el concepto de links entre puntos dato, para determinar cual es el par de clusters óptimo que se debe combinar en cada paso.
El concepto de link entre un par de puntos se define como la cantidad de vecinos en común que poseen los mismos.
Para determinar si un par de puntos son vecinos se utiliza una función de similitud (sim). Si el valor de esta aplicado a dos puntos excede un cierto valor dado, los puntos se consideran vecinos.

$$Sim(pi,pj) \leq \text{Umbral de Similitud}$$

donde A es un cierto valor que ingresa el usuario.
La función de similitud, en nuestro caso, corresponde a los diferentes criterios de distancia entre puntos descriptos en 2.1.

Este algoritmo, al igual que el anterior, es un algoritmo aglomerativo, la única diferencia es el criterio que se utiliza para determinar cuales son los dos clusters a combinar en cada paso. El primer algoritmo se basa en distancia entre clusters mientras que Rock se basa en el concepto de links, o cantidad de vecinos en común entre dos clusters.

3.3 Ingreso de datos de entrada y visualización de los resultados

El ingreso de los datos de entrada de los algoritmos, será mediante una interface gráfica donde el usuario puede clickear sobre una cuadrícula en pantalla para indicar la posición de cada punto dato, especificando a su vez si se trata de un cliente o un depósito, la producción o demanda del mismo (dependiendo del punto) así como también la ventana de tiempo y en el caso de los clientes el usuario debe ingresar la lista de los depósitos no compatibles. La visualización de los datos de entrada será como en la figura 6. Donde los depósitos se diferencian de los clientes pues son los puntos de mayor tamaño y en color azul. Mientras que los clientes son los puntos pequeños de color rojo. Para todos ellos se visualiza un número identificador.

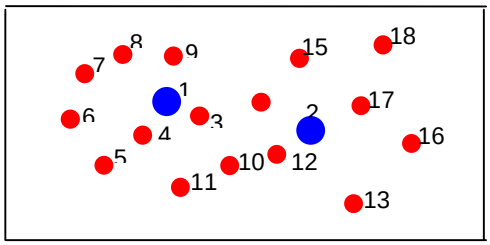


Figura 6

Otra opción es cargar un archivo de texto según el formato que se especifica en la sección 2.4.
Para la visualización de los resultados de los algoritmos, se presentan en pantalla cada uno de los puntos (clientes y depósitos) y para saber a que cluster pertenecen, se les asigna un color. Es decir, todos los puntos pertenecientes a un cluster tienen asignado el mismo color. Además se numeran los clusters obtenidos, de manera que todos los puntos pertenecientes un cluster tienen el mismo número. En la figura 7 se muestra un ejemplo de cómo podría ser la visualización del resultado de clustering para los datos de la figura 6.
Además de la interface gráfica, los resultados se guardan en un archivo de texto según el formato que se especifica en la Sección 2.4.

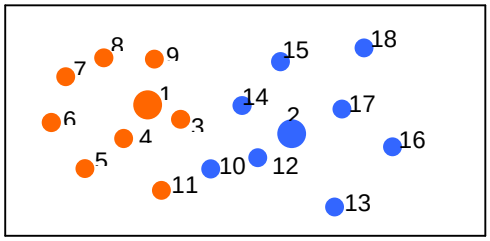


Figura 7

3.4 Formato Entrada/Salida de los algoritmos

Los datos/puntos de entrada tendrán asociados los siguientes datos:
- identificador del cliente o depósito (valor entero).
- coordenada x (valor real).
- coordenada y (valor real).
- demanda / producción según sea cliente o deposito (valor real).
- centro de la ventana de tiempo en minutos (valor entero).
- lista de depósitos incompatibles con el cliente (lista de enteros).

Para la entrada/salida de los algoritmos se utilizarán archivos de texto, cuyo formato será el siguiente:

ARCHIVO DE DATOS
N1
N2
IdDeposito1 x1 y1 t1 d1
IdDeposito2 x2 y2 t2 d2
.

.
IdCliente1 x3 y3 t3 p1 LNC <lista de identificadores de los depósitos no compatibles con el cliente IdCliente1> FinLNC
IdCliente2 x4 y4 t4 p2 LNC <lista de identificadores de los depósitos no compatibles con el cliente IdCliente2> FinLNC
.
.

Donde:
- N1 y N2 son enteros que indican la cantidad total de depósitos y clientes respectivamente.
- xi e yi son reales que indican las coordenadas.
- ti es un entero que indica el centro de la ventana de tiempo, medida en minutos transcurridos desde las 0hs.
- pi y di son reales

ARCHIVO DE RESULTADOS
Coeficiente wxy
Coeficiente wt
N1
N2
IdDepo , Nro_clientes, IdCli1, IdCli2,.....,IdClin
.
Donde:
- N1 y N2 son enteros que indican la cantidad total de depósitos y clientes respectivamente.
- Coeficiente xy, corresponde al valor real que se utilizó para ponderar la distancia euclídea entre dos puntos, en el caso que corresponda.
- Idem anterior para la diferencia entre las ventanas de tiempo, en el caso que corresponda.

3.5 Evaluación de resultados

Para evaluar los resultados se realizarán pruebas comparando resultados de clustering obtenidos usando distintas funciones de distancia en términos de criterios de bondad de clusters y también en términos del costo total del ruteo. Este último se mide como la suma de los largos de las rutas que resulta de rutear sobre una asignación de clientes a depósitos.

También se deberá comparar el costo de ruteo obtenido cuando la asignación se realiza mediante heurísticas de asignación [13] contra el costo del ruteo obtenido cuando la asignación se realiza con los algoritmos de clustering.

3.5.1 Algoritmo de ruteo

El algoritmo de ruteo es un adaptación del algoritmo Clark & Wright [14].

Consideramos a una ruta como completa, cuando no es posible incluir a otro cliente en ella.
Una ruta unitaria es una ruta formada por un solo cliente.

El algoritmo utiliza el concepto de ahorro (SAV, por “saving” en inglés) como medida del costo del ruteo. Se define el ahorro de incluir al cliente c en una ruta entre i,j como:

$$SAV_c\ (i,j) = 2d_{oc} – (d_{cj} + d_{ic} - d_{ij})$$

donde llamamos d_{ij} a la distancia entre i y j, o al depósito e i,j pueden ser clientes o el depósito.
Intuitivamente SAV_c (i,j) indica cuanto se ahorra en el costo total al insertar al cliente c entre i y j, en lugar de mantenerlo en una ruta unitaria. Un valor positivo de ahorro quiere decir que es conveniente eliminar la ruta unitaria de c y colocarlo entre i y j. Un valor negativo indica que es mejor mantener a c en la ruta unitaria. El SAV mide la diferencia entre las dos posibles configuraciones que se puede ver en la Figura 8. Estas configuraciones se diferencian por los arcos con línea continua.

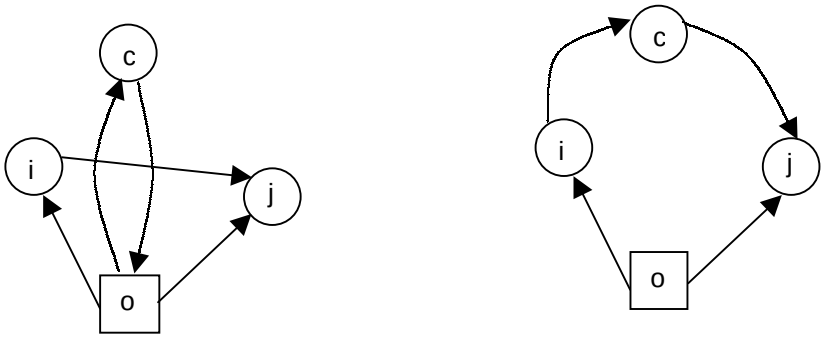


Figura 8

El seudocódigo es el siguiente:

```
procedure VRP()
begin
    while < queden rutas no completas y unitarias > and <Existe SAV_c (i,j) ≥ 0> do
      begin
        for <todo cliente c en una ruta unitaria> do
          begin
            <calcular SAV_c (i,j) ∀ i,j>;
          end
        <sea c / maximiza SAV_c (i,j), para una ruta r >;
        <colocar c en r entre i,j >;
      end
end
```

III. ANÁLISIS DE FUNCIONALIDAD

1. Objetivo

El objetivo de esta sección consiste en definir las funcionalidades a implementar para cumplir con los requerimientos definidos en el análisis de requerimientos.

2. Análisis

Básicamente las funcionalidades de la herramienta se pueden dividir en 4 grupos:

- Ingreso de los datos mediante una interface gráfica.
- Ejecución de algoritmos de clustering y heurísticas de asignación.
- Visualización de los resultados del clustering mediante interface gráfica.
- Visualización del costo de ruteo a partir de una asignación.

2.1 Ingreso de los datos

Consiste en la definición de clientes y depósitos mediante una interface gráfica.
La herramienta permitirá guardar el resultado en un archivo de texto con el formato especificado en el análisis de requerimientos.

Las funcionalidades para el ingreso de datos serán las siguientes:

- Creación de un nuevo archivo de datos.
- Modificación de un archivo de datos existente.

Tanto en la creación como en la modificación de un archivo de datos, se permitirá establecer para cada punto:

- Tipo de punto (cliente o depósito). Los depósitos se diferenciarán de los clientes por el tamaño y color del punto.
- Demanda o Producción dependiendo del tipo.
- Centro de la ventana de tiempo correspondiente al horario de servicio en formato hh:mm.
- Lista de depósitos incompatibles, en caso de que el punto sea de tipo cliente.

2.2 Ejecución de los algoritmos de clustering

Una vez creado o modificado un archivo de datos es posible seleccionar uno de los algoritmos disponibles AGLOMERACION o ROCK.

Para el algoritmo de AGLOMERACION se solicitará al usuario los siguientes parámetros:

- Distancia entre puntos.
- Distancia entre clusters.
- Ponderación para la distancia euclídea en el caso que la distancia entre puntos seleccionada sea una suma ponderada.
- Ponderación para la diferencia entre las ventanas de tiempo en el caso que la distancia entre puntos seleccionada sea una suma ponderada.

Para el algoritmo ROCK se solicitará al usuario los siguientes parámetros:

- Distancia entre puntos.
- Ponderación para la distancia euclídea en el caso que la distancia entre puntos seleccionada sea una suma ponderada.
- Ponderación para la diferencia entre las ventanas de tiempo en el caso que la distancia entre puntos seleccionada sea una suma ponderada.
- Función f(theta). Correspondiente a un valor real estimado, sabiendo que f(theta) cumple la propiedad de que cada punto que pertenece al cluster Ci tiene aproximadamente $n_i^{f(theta)}$ vecinos donde ni es la cantidad de puntos en el cluster Ci. Para este valor se presentará un valor sugerido por defecto.
- Umbral de similitud. Valor real que determina cuando dos puntos son vecinos de acuerdo a la formula sim(p1,p2) < UmbralSimilitud . Para este valor también se presentará un valor sugerido por defecto.

También permite indicar si se guarda las trazas generadas por los algoritmos. Estas contienen la información sobre que clusters se fueron formando en los pasos intermedios en los algoritmos.

Todos los archivos generados por los algoritmos se guardan en el mismo directorio donde se encuentra el archivo de datos. Los archivos generados tienen como prefijo el nombre del archivo de datos más un sufijo que indica el contenido.

El algoritmo también proporcionará información relativa al tiempo de ejecución y medidas de bondad

2.3 Salida y visualización gráfica de los resultados de clustering

Una vez finalizada la ejecución del algoritmo seleccionado automáticamente se visualizan los resultados en forma gráfica. Para identificar los distintos clusters se presenta cada cluster con un color diferente y con un número de cluster. De esta forma todos los puntos, clientes y depósito, pertenecientes a un mismo cluster se visualizan con un mismo color y con una etiqueta correspondiente al depósito al que fue asignado.

De forma opcional el usuario podrá visualizar la siguiente información en el mapa de puntos. Para los puntos correspondientes a clientes es posible visualizar los siguientes datos:
- Identificador del depósito al que fue asignado
- Identificador del cliente
- Lista de depósitos incompatibles.

Mientras que para los puntos correspondientes a depósitos es posible visualizar los siguientes datos:
- Identificador de depósito.
- Demanda restante. Corresponde a la demanda del depósito una vez que se han restado las producciones de los clientes asignados a él.

En la Figura 9 se puede ver como se visualizan los resultados de asignación obtenidos mediante los algoritmos de clustering. Se identifica cada cluster obtenido asignando un mismo color a todos los punto (clientes y depósito) que pertenecen al mismo cluster.

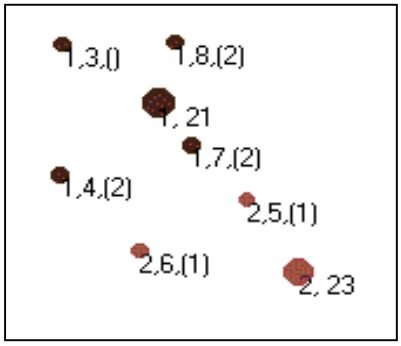


Figura 9

2.4 Ejecución de Heurísticas de asignación

Esta funcionalidad permite seleccionar una de las 4 heurísticas disponibles, ingresar los parámetros necesarios y aplicar el algoritmo obteniendo una asignación de clientes a depósitos.

Los parámetros necesarios son:
- Velocidad del vehículo.
- Constante escalar de distancia.

2.5 Visualización del costo de ruteo

Esta funcionalidad permite ingresar los parámetros necesarios para el ruteo y aplicar el ruteo con la asignación obtenida a partir de una asignación hallada previamente, ya sea con clustering o con las heurísticas [13], y visualizar el costo de ruteo y duración.
El algoritmo de ruteo es un adaptación del algoritmo Clark & Wright [14].

Los parámetros necesarios para el ruteo son:
- Capacidad del vehículo.
- Tiempo de espera máximo en ruta.
- Velocidad del vehículo.
- Constante escalar de distancia.

Una vez especificados estos parámetros se podrá ejecutar el algoritmo de ruteo y visualizar el costo de ruteo (suma del largo de las rutas).

IV. ADAPTACIÓN AL PROBLEMA

1. Objetivo

Se discutirá la factibilidad de aplicación de algoritmos de clustering aglomerativos, como algoritmo de asignación para el problema de ruteo con múltiples depósitos y ventanas de tiempo. El problema original es un problema de recolección con múltiples depósitos y ventanas de tiempo.

2. Análisis

Se plantean las diferentes alternativas para la aplicación de los algoritmos: Aglomeración y Rock.

Ambos son algoritmos aglomerativos, la única diferencia consiste en el criterio que se utiliza para determinar cuales son los dos clusters óptimos a combinar en cada paso del algoritmo.
Se busca maximizar o minimizar cierta función objetivo. En el caso tradicional de aglomeración, se utiliza como función objetivo alguna de las distintas medidas de distancia entre clusters (Ver Análisis de requerimientos), y en este caso se desea encontrar el par de clusters que minimizan tal función. En el caso del Rock, se utiliza como función objetivo, la cantidad de vecinos en común entre dos clusters, y en este caso se desea encontrar el par de clusters que maximizan esta función.

Las variantes que se discuten a continuación, son válidas para los dos algoritmos, teniendo en cuenta que optimizar la función objetivo implica:
- Minimizar la distancia entre clusters, en el caso del algoritmo de aglomeración tradicional.
- Maximizar la cantidad de vecinos en común, en el caso del algoritmo Rock.

2.1 Consideraciones para aplicar algoritmos aglomerativos

Básicamente, estos algoritmos constan de los siguientes pasos:
Paso 1)
Se comienza con un conjunto inicial de clusters. Todos los puntos (clientes y depósitos) deberán estar contenidos en alguno de estos clusters.
Paso 2)
Se halla el par de clusters óptimo tal que al agruparlos minimizan la función de distancia entre clusters. Esta función dependerá de la posición y la ventana de tiempo. Se combinan estos dos clusters formando uno nuevo.
Paso 3)
Si, la cantidad de clusters cumple la condición de parada, el proceso termina. En caso contrario, se vuelve al paso 2.

En el caso que se desee conocer las decisiones tomadas en cada paso del algoritmo, se puede mantener un histórico de las combinaciones de clusters realizadas.

2.1.1 Conjunto de clusters de partida

Se presentan dos alternativas, como conjuntos de clusters de partida:

- Comenzar con una cantidad de clusters igual a la cantidad de clientes más la cantidad de depósitos, donde cada uno de los clusters contiene un solo elemento (un cliente o un depósito). Combinar los clusters siempre que contengan 0 o 1 depósito. Terminar cuando tenemos tantos clusters como depósitos. Puede ocurrir que queden depósitos sin ningún cliente asignado, así como también clientes que no se asignen a ningún cluster.
- Dados el conjunto de depósitos y el conjunto de los clientes. Sea k la cardinalidad del conjunto de depósitos. Comenzar con una cantidad de clusters igual a la cantidad de clientes, donde cada cluster contiene un solo elemento (un cliente) y aplicar aglomeración hasta obtener k clusters. Una vez obtenidos los k clusters, se agregan k clusters adicionales donde cada uno de ellos contiene un solo elemento (un depósito), y se continua con el proceso de aglomeración con la condición de combinar un cluster que contiene un depósito con otro que no tiene ninguno. Terminar cuando todos los depósitos han sido asignados a un cluster de clientes o no puedan haber más combinaciones. Puede ocurrir que queden depósitos sin ningún cliente asignado, así como también clientes que no se asignen a ningún cluster.

2.1.2 Combinación de clusters

Para que se puedan combinar los dos clusters que optimizan la función objetivo, se deben cumplir las siguientes condiciones:

- A lo sumo uno de los clusters contiene un depósito.
- En el caso que uno de los clusters contenga un depósito, la suma de la producción de los clientes de ambos clusters no debe superar la demanda total del depósito.
- No se pueden asignar clientes a depósitos incompatibles.

Estas condiciones alteran el algoritmo original de aglomeración. Por lo cual, se exponen diferentes alternativas para el tratamiento de estas restricciones.

Restricciones por la demanda de los depósitos

Dados dos clusters C y D óptimos a combinar según la función objetivo, donde C es un cluster que contiene solo clientes y D contiene al menos un depósito. Puede darse el caso que la producción total supere la demanda. Para resolver esto tenemos dos alternativas:

- Dividir el cluster C y recalcular óptimos. Quitar sucesivamente el cliente de máxima producción de C mientras la producción total siga superando a la demanda del depósito en D. Crear un nuevo cluster con estos clientes. Recalcular los dos clusters óptimos a combinar.
- Evitar el caso. No combinar un cluster de solo clientes con otro cluster de solo clientes si se cumple que la producción total supera la demanda restante de cualquiera de los depósitos. La demanda restante se calcula restando a la demanda del depósito las producciones de los clientes ya asignados.

La primera alternativa implica destruir clusters previamente formados por el algoritmo y el recálculo de los dos clusters óptimos a combinar, perdiendo tiempo en cálculos que luego pueden desecharse. Una mejor solución, es la segunda opción, ya que evita que se formen clusters de solo clientes que luego no se puedan combinar, debido a las restricciones de la demanda, con ningún otro cluster. Esto asegura que no habrán restricciones por demanda al momento de combinar un cluster de solo clientes con uno que ya contiene un depósito.

Restricciones por incompatibilidad cliente-depósito

Dados dos cluster C y D óptimos a combinar según la función objetivo, donde C es un cluster que contiene solo clientes y D contiene al menos un depósito. Puede darse el caso que no todos los clientes en C sean compatibles con el depósito en D.
Para resolver esto tenemos dos alternativas:

- Dividir el cluster C y recalcular óptimos. Quitar de C todos los clientes incompatibles con el depósito en D. Crear un nuevo cluster con estos clientes. Recalcular los dos clusters óptimos a combinar.
- Evitar el caso. Solo combinar un cluster de solo clientes con otro cluster de solo clientes si se cumple que todos son compatibles a los mismos depósitos.

La primera alternativa implica destruir clusters previamente formados por el algoritmo y el recálculo de los dos clusters óptimos a combinar. Una mejor solución, es la segunda opción, ya que evita que se formen clusters de solo clientes que luego no se puedan combinar debido a la incompatibilidad (evitando desperdiciar cálculos realizados en pasos anteriores). Esto asegura que no habrán restricciones debido a clientes incompatibles al momento de combinar un cluster de solo clientes con uno que ya contiene un depósito.

2.1.3 Criterio de parada

El proceso de aglomeración continuará mientras la cantidad de clusters sea mayor a la cantidad de depósitos o mientras se puedan combinar algún par de clusters. El resultado final será un conjunto de clusters, donde cada uno de ellos contendrá un subconjunto de los clientes y uno y solo uno de los depósitos. Además, se cumplirá que: la producción total de los clientes asignados a un cluster no supera la demanda del depósito correspondiente.

2.2 Consideraciones para aplicar el algoritmo Rock

Para decidir cuál es el par de clusters óptimo a combinar en cada paso del algoritmo, se usa una función que se basa en la cantidad de vecinos en común entre los clusters. Esta función es la siguiente:

$$g(C_i, C_j) = \frac{Link\ [C_i, C_j]}{(n_i + n_j)^{1+2f(\theta)} - n_i^{1+2f(\theta)} - n_j^{1+2f(\theta)}}$$

Donde Link [Ci, Cj] es el número de links cruzados entre los clusters Ci y Cj.

$$Link\ [Ci, Cj] = \sum_{p_q \in Ci,\ p_r \in Cj} link\ (p_q, p_r)$$

Recordar que el link entre un par de puntos se define como la cantidad de vecinos en común, y para determinar si dos puntos son vecinos se utiliza la función $sim(p_i,p_j)$. Si su valor no supera cierto límite dado por el usuario, decimos que p_i y p_j son vecinos.

Se utilizará la función $sim(p_i,p_j)$ normalizada, es decir, que toma valores entre 0 y 1. Su valor se obtiene calculando la distancia entre el par de puntos p_i y p_j (utilizando alguno de los criterios de distancia entre puntos. Ver Sección II Análisis de requerimientos) y dividiendo ese valor por la distancia máxima entre dos puntos dato.

La función $f(\theta)$ es tal que cumple:

Cantidad de vecinos esperados de $p_i = ni^{f(\theta)}$, con ni = |Ci| y pi $\in$ Ci

Hallar el valor $f(\theta)$ no es tarea fácil, por lo que es mas conveniente utilizar una aproximación a su valor.

Una posibilidad para obtener una estimación es ejecutar el algoritmo de aglomeración tradicional y en base a los clusters obtenidos, se calcula la cantidad promedio de vecinos por cada cluster y de allí se puede obtener un valor aproximado de $f(\theta)$. Ver punto 2.2.2

2.2.1 La función objetivo

Se busca maximizar la cantidad de vecinos en común entre puntos de un mismo cluster, por lo tanto la función obvia que se busca maximizar es:

$$\sum_{i=1,k} \sum_{p_q,p_r \in C_i} link(p_q,p_r)$$

recordar que $link(p_q,p_r)$ es la cantidad de vecinos en común que tienen p_q y p_r.
Ci representa al cluster i, y k es la cantidad total de clusters.

Sin embargo, esta formula (ver [6]) no fuerza a que puntos con pocos links se dividan en distintos clusters. Para remediar esto es que dividimos la cantidad total de links entre dos puntos en el cluster C_i por la cantidad total esperada de links en C_i y luego ponderando por n_i que es el número de puntos en C_i [6].

$$E_l = \sum_{i=1,k} n_i \sum_{p_q,p_r \in C_i} link(p_q,p_r) / n_i^{1+2f(\theta)}$$

recordar que $link(p_q,p_r)$ es la cantidad de vecinos en común que tienen p_q y p_r.
Ci representa al cluster i, y k es la cantidad total de clusters.

Donde f(θ) es una función que depende del conjunto de datos y del tipo de clusters en que estamos interesados. Además cumple la propiedad de que cada punto que pertenece al cluster C_i tiene aproximadamente $n_i^{f(\theta)}$ vecinos en C_i.
Si tal función f existe, entonces, puesto que podemos asumir que puntos fuera de C_i resultan en una cantidad pequeña de links con los puntos de C_i, cada punto en C_i contribuye con $n_i^{2f(\theta)}$ links (uno por cada par de sus vecinos). De esta forma obtenemos $n_i^{1+2f(\theta)}$ como el número esperado de links entre pares de puntos en C_i.

Puesto que la meta es encontrar los clusters que maximizan E_l , usamos una medida similar para determinar el mejor par de clusters para combinar en cada paso.
Para C_i , C_j un par de clusters, sea:

Link[C_i , C_j] el número de links cruzados entre C_i y C_j.
Esto es:

$$\text{Link}[C_i, C_j] = \sum_{\substack{p_q \in C_i \\ p_r \in C_j}} \text{link}(p_q, p_r)$$

Entonces, se define la medida $g(C_i , C_j)$ para combinar C_i con C_j como [6]:

$$g(C_i, C_j) = \frac{\text{Link}[C_i, C_j]}{(n_i + n_j)^{1+2f(\theta)} - n_i^{1+2f(\theta)} - n_j^{1+2f(\theta)}}$$

El par de clusters para el cual $g(C_i, C_j)$ es máximo, es el mejor par de clusters ha ser combinado en cada paso [6].

2.2.2 Estimación de f(θ)

Por supuesto, determinar un valor exacto para f(θ) es difícil. Sin embargo, encontramos que si los clusters están relativamente bien definidos, aún un valor impreciso pero razonable de f(θ) funciona bien en la práctica. Además, los errores introducidos en la estimación de f(θ) afectan a todos los clusters por igual, y no penaliza excesivamente un cluster con respecto a los otros.

Para estimar f(θ), se puede utilizar la siguiente estrategia:

- Se aplica el algoritmo aglomerativo tradicional, obteniendo un conjunto de clusters.
- Por cada cluster Ci obtenido, calcular:
- Para cada punto pi ∈ Ci, la cantidad de vecinos que tiene en Ci.
- Hallar el promedio de la cantidad de vecinos en Ci. (la suma de los valores hallados en el paso anterior, dividido la cantidad de puntos en el cluster).
- Hallar el promedio de los valores hallados en el punto 2) para cada cluster.

2.3 Conclusiones

Es factible aplicar ambos algoritmos, tomando las consideraciones descriptas para adecuar los algoritmos a las condiciones del problema.

Sería conveniente tener categorías de depósitos. Donde cada cliente pueda pertenecer a una o varias de estas categorías.

De esta forma se aplicaría clustering a cada categoría de depósitos y sus clientes compatibles evitando las restricciones por incompatibilidad cliente - depósito.
El resultado sería un conjunto de clusters para cada categoría.

La ventaja de tener categorías de depósitos, es que al momento de hallar los clusters óptimos a combinar influye en mayor medida la distancia y la ventana de tiempo y se liberan la restricciones de incompatibilidad entre clientes y depósitos.

En principio, no se utilizarán categorías de depósitos. Esto puede llevar a soluciones pobres si los clientes tienen un conjunto de depósitos no compatibles muy distinto con respecto a los conjuntos de depósitos de los demás clientes. En este caso, podría ocurrir que los algoritmos de aglomeración realicen pocas combinaciones antes de terminar. Lo que produciría que los algoritmos se deformen en un algoritmo de tipo K-Means [1], recordemos que este algoritmo asigna clientes a clusters mientras que la idea de los algoritmos de aglomeración es realizar la combinación clusters previamente formados.

Es importante trabajar en una escala en común para medir las coordenadas x,y,t ya que el resultado del clustering se puede ver muy afectado por las diferencias entre las escalas en que se mide cada dimensión. Por esta razón las coordenadas se normalizan llevándolas al intervalo [0,1].

V. DISEÑO

1. Objetivo

En esta sección se presenta el seudocódigo los algoritmos a implementar y el diseño de los módulos que identificaron para la implementación de los algoritmos de clustering: AGLOMERACION y ROCK.

2. Seudocódigo

Se presenta el seudocódigo de los algoritmos ROCK y AGLOMERACION

2.1 Algoritmo ROCK

El procedimiento Cluster que se describe corresponde al seudocódigo del algoritmo Rock (ver [6]).

```
Procedure Cluster (S, k)
        Begin
1               link := compute_links(S)
2               For each s ∈ S do
3                       q[s] := build_local_heap(link, s)
4               Q := build_global_heap(S, q)
5               While size(Q) > k do {
6                       u := extract_max(Q)
7                       v := max(q[u])
8                       delete(Q, v)
9                       w := merge(u, v)
10                      For each x ∈ q[u] ∪ q[v] do {
11                  link[x,w] := link[x,u] + link[x, v]
12                  delete(q[x], u); delete(q[x], v)
13                  insert(q[x], w, g(x,w)); insert(q[w], x, g(x,w))
14                          update(Q, x, q[x])
15                    }
16                    insert(Q, w, q[w])
17                    deallocate(q[u]), deallocate(q[v])
18              }
End
```

El parámetro **S** representa el conjunto de datos a clusterizar y el parámetro **k** la cantidad de clusters deseados, que en nuestro caso será siempre a la cantidad de depósitos.
El procedimiento comienza calculando el número de links entre pares de puntos en el paso 1. Inicialmente cada punto es un cluster separado. Para cada cluster i construimos un heap local q[i]. El heap local q[i] contiene todo cluster j tal que:

$$Link(i,j) \neq 0$$

Los clusters j en q[i] están ordenados en orden decreciente de g(i,j) con respecto a i. Es decir, que para cada cluster tengo un heap, q[i] de clusters tales que link(i,j) $\neq$ 0, ordenados decrecientemente según g.

Además del heap local q[i] para cada cluster i, el algoritmo mantiene un heap global, Q, que contiene todos los clusters, los cuales están ordenados en forma decreciente según el valor de g. De este modo, g(j,max(q[j])) se usa para ordenar los clusters j en Q, donde max(q[j]) es el mejor cluster a combinar con el cluster j.
En cada paso el max cluster j en Q y el max cluster en q[j] es el mejor par de clusters a combinar.
El loop While, en el paso 5 itera hasta que en Q queden k clusters. Además también para si el número de links entre todo par de los clusters restantes es 0.
En cada paso, del loop while, se extrae el max cluster u de Q por extract_max y q[u] se usa para determinar el mejor cluster v para u.
Puesto que los clusters v y u serán combinados, las entradas para v y u en Q se borran. Los clusters v y u se combinan entonces en el paso 9 para crear w que contiene |u|+|v| puntos.

Hay dos tareas que deben realizarse cuando dos clusters se combinan.
- Para cada cluster que contiene a u o v en su heap local, los elementos u y v deben ser reemplazados por w y el heap local debe ser actualizado.
- Debe crearse un nuevo heap local para w.

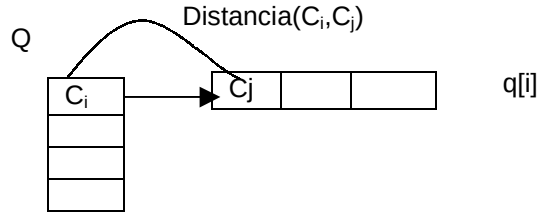
Estas dos tareas se realizan en el loop for del paso 10 al paso 15.

El número de links entre los clusters x y w es simplemente la suma del número de links entre x y u, y x y v. Esto se usa para calcular g(x,w) y los dos clusters son insertados cada uno en los otros heaps locales.
Notar que q[w] solo puede contener clusters que previamente estaban en q[u] o en q[v] puesto que estos son los únicos clusters que no tiene cero links con el cluster w. Notar además que, como resultado de combinar u y v, es posible que el cluster u o v fuera previamente el mejor para combinar con x y ahora w se vuelve el mejor para ser combinado. Además es posible que ninguno, ni u ni v fueran el mejor cluster a ser combinado con x, pero ahora w es el mejor para combinarse con x. Para tales casos, siempre que el max cluster en el heap local de x cambia, el algoritmo necesita relocalizar x en Q para reflejar la información relacionada al nuevo mejor cluster para x (14). El procedimiento también necesita asegurarse que Q contiene el mejor cluster ha ser combinado para el cluster w.

```
Procedure Compute_links(S)
Begin
        Compute nbrlist[i] for every point i in S
        Set link[i,j] to be zero for all i,j
        for i = 1 to n do {
            N := nbrlist[i]
            for j = 1 to |N| -1 do
                for l = j +1 to |N| do
                    link[N[j] , N[l]] = link[N[j] , N[l]] +1
        }
End
```

El algoritmo compute_links, para cada punto, luego de hallar su lista de vecinos, considera todos los pares de sus vecinos. Para cada par, el punto contribuye con un link. Si el proceso se repite para cada punto y la cuenta de links se incrementan para cada par de vecinos entonces al final se obtienen las cuentas de links para todo para de puntos.

2.2 Algoritmo AGLOMERACION



Nodo de heap Global
- Identificador de cluster
- Puntero a cluster
- Heap local

Nodo Heap Local
- Identificador de cluster
- Puntero a cluster
- Distancia con el cluster del nodo de heap global.

```
Procedure Aglomeracion(file FileName, TipoDistanciaCluster dc,
        TipoDistanciaPunto dp)
HeapGlobal Q;
HeapLocal q;
Begin
      // carga Heaps iniciales con un cluster por punto dato
        lista_puntos= CargarListaPuntosDato(FileName, Q,dc,dp, K);
        HuboCambios = True;
        Mientras Q.CantElem > K hacer and HuboCambios
              (idC1,idC2) = DarIdOptimos(Existen);
              if Not Existen then
                    HuboCambios = false
              Break Mientras
              Endif
              // combinar los optimos
              CRes= Combinar(Q.Min, Q.min.HeapLocal.Min);
              Q.Eliminar(IdC1);
              Q.Eliminar(IdC2)

              // agrega a Q y calcula todo el heap local. (*)
              Q.Agregar(Cres);
              IdCRes = Cres.DarIdentificador;
              //Actualiza los heaps locales
              For each c in Q
                    If c.Identificador <> IdCRes then  /* pues ya se calculo en (*)
                          /*recorro heaps locales
                          q =DarHeapLocal(c)
                          q.Eliminar(idC1);
                          q.Modificar(idC2,IdCres);
                    endif
              Fin For
        Fin Mientras
End

Procedure CargarListaPuntosDato(file filename, HeapGlobal Q, TipoDistanciaCluster dc,
          TipoDistanciaPuntos dp, Int NdepoTot )
Cluster C;
PuntoDato p;
Begin
        Ndepot =LeerCantidadDepositos();
        Nclientes = LeerCantidadClientes();
        Para cada punto p especificado en el archivo
              p.CrearPuntoDato(x,y,t,pd,lnc); / crear el punto dato
              C.CrearVacio();
              C.Agregar({p});  /* contiene un solo elemento*/

              // en la raíz del heap queda el de mínima distancia
              Q.Agregar(C, Ascendente)
              Para cada cluster c en Q
                    q = DarHeapLocal(c);
                    NodoLocal.Identificador = C.Identificador
                    NodoLocal.PtrCluster = &C;
                    NodoLocal.Distancia = Distancia (c,C,dc,dp)
                    q.Agregar(NodoLocal);
              Fin Para
              Q.HeapLocal = q
      Fin Para
End
```

Los heaps locales mantienen la información sobre los clusters factibles de combinarse con el cluster correspondiente en el heap global. En estos heaps locales se encontraran aquellos clusters que cumplen (cluster del nodo global y cluster de nodo local correspondiente)

- Tienen idéntica lista de incompatibles
- A lo sumo uno de ellos tiene deposito.
- Si uno de ellos tiene deposito, entonces la producción total de ambos clusters no sobrepasa la demanda de tal deposito.

No tiene sentido guardar la información relativa a otros clusters en los heaps locales, puesto que de todas formas no se van a seleccionar para combinarse.
Con esto se evita el cálculo costoso de distancias entre dos clusters que son imposibles de combinar debido a las restricciones impuestas al problema.

3. Diseño de módulos

En este punto se describen los módulos identificados, de los cuales se detallan sus interfaces y sus dependencias con otros módulos.

Se ha optado por un diseño orientado a objetos, donde se identifican las clases que implementan las estructuras auxiliares necesarias para la ejecución de los algoritmos, los algoritmos, y la interface para el ingreso de datos y el almacenamiento de los resultados.

Para el ingreso de los datos de entrada, la visualización y almacenamiento de los resultados, se identifican los siguientes objetos:

- Mapa de puntos (para ingreso de datos y visualización de resultados a través de una interface gráfica)
- Manejo de Archivos de datos de Entrada/Salida

Como estructuras auxiliares para la implementación de los algoritmos se identifican los siguientes módulos:

- Punto Dato. Guarda la información de los puntos dato.
- Cluster. Conjunto de puntos asignados a un cluster.
- Lista. Lista de depósitos incompatibles.
- Heap global
- Heaps Locales

3.1 Definición de Tipos

Se requieren los siguientes tipos de datos.

tipo_punto = (cliente, deposito)
tipo_orden = (ascendente, descendente)
listaNC = lista (*Integer*)
lista_puntos = lista (*punto_dato*)
tipo_dist_punto = (suma_ponderada, cuadra, chebychev, angulo)
tipo_dist_cluster = (single, complete, upgmc, wpgmc, upgma, wpgma, within_groups, ward)
tipo_pos_heap = Integer
Algoritmo = (AGLOMERACION, ROCK)

3.2 Definición de Módulos

Se dispone de un módulo *lista* que implementa una lista genérica de elementos de tipo T Este tipo se podrá instanciar para obtener listas de diferentes tipos de objetos.
La lista consta de un conjunto finito de elementos de tipo T genérico y apuntadores que permiten su recorrido ordenado. La lista posee dos apuntadores de posición uno que indica la posición actual en el recorrido de la lista más un apuntador auxiliar que permite el recorrido anidado de la lista.

generic module **lista** (T)
uses
exports
Procedure **CrearVacia** ();
/* Constructor de la clase.

Procedure **EsVacia** (**b**: out boolean);
/* Devuelve true si la lista es vacía y false en caso contrario.

Procedure **Agregar** (**elem**: in T);
/* Agrega el elemento elem a la lista.

Procedure **Borrar** (**elem**: in T);
/*Elimina el elemento elem de la lista si este pertenece a ella.

Function **Buscar**(**e**: T, **SetearPos**: in boolean, **res**:out boolean)
/* busca el elemento e en la lista. Si SetearPos tiene valor verdadero y el elemento e pertenece a la lista, la posicion de la lista apunta al elemento e. Si el elemento pertenece la función devuelve res con valor verdadero de lo contrario con valor falso.

Procedure **BorrarApuntado** (**elem**: in T);
Precondicion: El elemento apuntado existe.
/*Elimina el elemento de la posición actual de la lista.

Procedure **Primero** (**t**: out T);
Precondición: la lista no debe ser vacía
/*Devuelve posición el apuntador al primer elemento de la lista.

Procedure **Siguiente** (**t**: out T, **b**: out boolean);
Precondición: la lista no debe ser vacía.
/*La posicion actual de la lista pasa a ser el elemento siguiente al último accedido. Además, si existe tal elemento se devuelve b en true.

Procedure **Concatenar**(**l1** : in/out Lista(T), **l2** : in Lista(T));
/* Concatena l2 a l1. El resultado es que l1 es el resultado de concatenar ambas luego de la concatenación l1 continua siendo ordenada

Procedure **GuardarPosicion**()
/*guarda la posicion actual.

Procedure **RecuperarPosicion**()
/* recupera la posicion de la ultima vez que se ejecuto GuardarPosicion.
End

Se dispone de un módulo heap que implementa una cola de prioridad genérica que contiene elementos de tipo T. Se podrá instanciar, siempre y cuando el tipo tenga definida una función de orden

generic module ***heap*** *(T)*
uses tipo_orden
exports
Procedure **SetearOrden**(t:in TipoOrden)
Precondición: el heap no tiene elementos.
/*Setea el orden que debe mantener el heap. Puede ser ascendente o descendente según se desee tener en el tope del heap el valor menor o mayor respectivamente. x

Procedure **CrearVacio** (tam: in integer):
/* Constructor de clase. Crea un heap vacío con capacidad máxima de tam elementos.

Procedure **CambiarTam**(**tam**: in integer)
/* Modifica el tamaño del heap. Crea un heap vacío con capacidad máxima de tam elementos.

Procedure **Agrega** (**t:** in T, **orden**: in tipo_orden);
/* Agrega el elemento t al heap de acuerdo al orden dado

Procedure **SuprimeMinMax** (**t**: out T);
/* Elimina y devuelve el elemento mayor/menor del heap, dependiendo si el orden del heap es descendente/ascendente.

Procedure **Modificar**(p:in tipo_pos_heap)
/* Modifica el valor del elemento de la posición p, reorganizando el heap si es necesario.

Procedure **Eliminar**(**n**:in Nodo)
/* Elimina el nodo n, reorganizando el heap si es necesario.

Function **Buscar**(**t**: in T): tipo_pos_heap
// devuelve la posición del elemento t si este pertenece al heap. Si no lo encuentra
// devuelve –1

Function **DarCantElems**(): Integer
// Devuelve la cantidad de elementos en el heap.

Procedure **DarElemento**(**p**: tipo_pos_heap,**e**: out T)
Precondición: el elemento de la posicion p existe.
// Devuelve en e el elemento de la posición p.

Function **DarTamanio**():Integer
// Devuelve la capacidad máxima que puede contener el heap.

Function **EstaLleno**():boolean
// Devuelve verdadero si se alcanzó la capacidad máxima del heap. Falso en otro caso.

Function **EstaVacio**():boolean
// Devuelve verdadero si el heap no contiene ningún elemento.

End

module **punto_dato**
uses integer, listaNC, tipo_punto
exports
Procedure **Crear** (**x**: in float, **y**: in float, **t**: in float, **tipo**: tipo_punto, **ProdDem**: in float,
l: in listaNC, id: in Integer);
/* Constructor de la clase

Procedure **Ubicación** (**x**: out float, **y**: out float):
/* Devuelve las coordenadas del punto_dato

Procedure **Ventana** (**t**: out float);
/* Devuelve la ventana de tiempo del punto_dato

Procedure **Tipo** (**tipo**: out tipo_punto);
/* Devuelve el tipo del punto_dato

Procedure **DemandaProduccion** (**demprod**: out float);
/* Devuelve la demanda/producción según corresponda

Procedure **EsCompatible** (**IdDepo**: in integer, **res**: out boolean);
Precondición: el punto_dato debe ser de tipo cliente
/*Devuelve true si el cliente es compatible con el depósito dado.

Function **DarIdentificador**(id:out Integer)
/* Devuelve el identificador del punto
End

module **cluster**
uses punto_dato, lista_puntos
exports
Procedure **CrearVacio** ();
/* Constructor de la clase

Procedure **Combinar** (**clus1**: in cluster, **clus2**: in cluster);
/* Crea un cluster nuevo a partir de los elementos de clus1 y clus2

Procedure **Agregar** (**l:** in lista_puntos)
Precondición: la lista de puntos debe contener solo puntos_dato de tipo cliente, si el cluster ya posee un punto_dato de tipo deposito
/* Agrega los puntos de l al cluster

Procedure **TieneDeposito** (**dep**: out boolean);
/* Devuelve true si el cluster contiene un punto_dato que es de tipo deposito

Procedure **DarCentro** (**x**: out float, **y**: out float, **t**: out float)
/*Devuelve la posición del centro del cluster

Procedure **DarProduccion** (**p**: out float)
/* Devuelve la suma de las producciones de todos los puntos de tipo cliente que pertenecen al cluster

Procedure **DarDemandaRestante (d**: out float)
Precondición: el cluster debe tener un punto_dato de tipo deposito.
/* Devuelve la demanda del depósito menos la producción total de los clientes del cluster.

Function **EsCompatible**(**idDepo**:in Integer,**res**:out boolean);
// devuelve res en true si todos los puntos del cluster son compatibles
// con el deposito idDepo

Function **DarDeposito**(**id**:out)
Precondición: el cluster tiene deposito asignado.
/*devuelve el identificador del deposito asignado al cluster

Procedure **DarListaPuntos**(**l**:out lista(punto_dato))
// devuelve en l la lista de puntos que forman el cluster

Procedure **DarLNC**(**l**:out lista(Integer))
// devuelve la lista de identificadores de depósitos incompatibles al cluster.

End

module **distancia**
uses tipo_dist_punto, tipo_dist_cluster, distancia_punto, distancia_cluster
exports

Procedure **Distancia** (**Id1**: in float, **Id2**: in integer, **d**: out integer, **tipo** : in tipo_distancia, wxy: in float, wt: in float, td: in tipo_dist_punto, tc: in tipo_dist_cluster);
end

module **distancia_punto**
uses tipo_distancia
exports
Procedure **SumaPonderada** (**p1**: in punto_dato, **p2**: in punto_dato, **wxy**: in float, **wt**: in float,**d**: out float);
Procedure **CityBlock** (**Id1**: in integer, **Id2**: in integer, **d**: out float);
Procedure **Chebychev** (**Id1**: in integer, **Id2**: in integer, **d**: out float);
Procedure **Angulo** (**Id1**: in integer, **Id2**: in integer, **d**: out float);
End

module **distancia_cluster**
uses tipo_distancia, tipo_similitud
exports
Procedure **SingleLinkage** (**Id1**: in integer, **Id2**: in integer, **tdist**: in tipo_distancia_punto)
Procedure **CompleteLinkage** (**Id1**: in integer, **Id2**: in integer, **tdist**: in tipo_distancia_punto)
Procedure **CentrosNoPonderados** (**Id1**: in integer, **Id2**: in integer, **tdist**: in tipo_distancia_punto)
Procedure **CentrosPonderados** (**Id1**: in integer, **Id2**: in integer, **tdist**: in tipo_distancia_punto)
Procedure **PromParesNoPonderados** (**Id1**: in integer, **Id2**: in integer,
tdist: in tipo_distancia_punto)
Procedure **PromParesPonderados** (**Id1**: in integer, **Id2**: in integer, **tdist**: in tipo_distancia_punto)
End

module **Algoritmos**
uses tipo_punto, tipo_orden, listaNC, lista_puntos, tipo_dist_punto, tipo_dist_cluster , tipo_pos_heap, lista (T), heap (T), punto_dato, cluster, distancia, distancia_punto, distancia_cluster
exports

Procedure **Aglomeracion** (**alg**: in Algoritmo, **tdc**: in tipo_distancia_cluster, **tdp**: tipo_distancia_punto, **NombreArchivoIn**: in string, **NombreArchivoOut**: in string, **NombreArchivoOutAux**: in string, **NombreArchivoHeap**: in string, **NombreArchivoClusters**: in string, **NombreArchivoMatrizLinks:** in string, **Intra**: out float, **Inter**: out float, **wxy**: in float, **wt**: in float, **DejarTrazas**: in boolean, **UmbralSimilitud**: in float, **ftheta**: in float)
// Ejecuta el algoritmo AGLOMERACION o ROCK, dependiendo del parámetro alg.
// Los parámetros wxy y wt se usan solo si la distancia entre puntos es suma ponderada.
// En NombreArchivoOut quedan el archivo de resultados.
// En NombreArchivoOutAux quedan los puntos que no fueron asignados y los valores de distancia
// intra e intercluster de la solución obtenida.
// En NombreArchivoClusters quedan los sucesivos clusters que se formaron durante
// la ejecución del algoritmo. Este archivo solo se genera en el caso que DejarTrazas
// sea verdadero.
// NombreArchivoHeap queda el contenido de los heaps, estructuras auxiliares, que
// determina que clusters se combinan en cada paso. Este archivo solo se genera en
// el caso que DejarTrazas sea verdadero.
// NombreArchivoMatrizLinks, quede el contenido de la matriz de links que se usa
// para el algoritmo ROCK. Este archivo solo se genera en el caso que DejarTrazas sea verdadero.
// Los parámetros UmbralSimilitud y ftheta solo se usan en el algoritmo ROCK.

Procedure **ArmarHeapsLocalesIniciales** (**H**: in/out Heap(T), **NuevoCluster**: in cluster, **tdc**: in tipo_distancia_cluster, **tdp**: in tipo_distancia_punto, **ListaClusters**: in/out lista(cluster), **wxy:** in float, **wt**: in float, **MatrizLinks**[ ][ ]: in/out integer, **Puntos**: conj_puntos, **ftheta:** in float)
// Agrega un nodo al heap global H correspondiente al NuevoCluster. Actualiza los
// heaps locales de los demás clusters.
End

module **IOPuntosDato**
uses tipo_punto, tipo_orden, listaNC, lista_puntos, tipo_dist_punto, tipo_dist_cluster , tipo_pos_heap, lista (T), heap (T), punto_dato, cluster, distancia, distancia_punto, distancia_cluster, Algoritmos
exports
Function **CargoPuntos**(**Alg**: in Algoritmo, **filenamedatos**: in string, **filenamelog**: in string, **tdp**: in string, **tdc**: in string, **DejarTraza**: in boolean);
// inicializa estructuras auxiliares con los puntos leidos desde el archivo
// creando un cluster por punto en el caso del algoritmo AGLOMERACION
// Para el algoritmo ROCK, solo se carga en un buffer todos los puntos
// en orden de aparicion en el archivo de datos para luego armar las estructuras
// auxiliares necesarias.
// si DejarTraza es verdadero deja un log por cada cluster creado en el archivo
// filenamelog.

Function **EscribirResultados(**filenameout**: in string,** filenamenoasignados**: in string);**
// escribe los resultados del clustering en el archivo filenameout
// con el formato especificado en el analisis de requerimientos para el
// archivo de salida.
End

VI. IMPLEMENTACIÓN

1. Objetivo

El objetivo de esta sección es describir cuales han sido las decisiones tomadas al momento de implementar la solución. Para ello se detallan, por ejemplo, las herramientas utilizadas para la implementación de los algoritmos de clustering, y por otro lado, las razones por las cuales se optó por una u otra herramienta.

2. Análisis

Entre las decisiones tomadas para la implementación de la solución, se encuentra la elección de la plataforma sobre la cual trabajar, los lenguajes de programación y las estructuras de datos mas convenientes para la implementación de las estructuras auxiliares.

2.1 Lenguajes de programación

El diseño orientado a objetos, y los tiempos de ejecución de los algoritmos a implementar, requieren un lenguaje orientado a objetos que sea muy eficiente. Esto nos lleva a seleccionar el lenguaje C++ para la implementación de tales algoritmos. Teniendo, además, la posibilidad de acceder a las herramientas Microsoft, se seleccionó Visual C++ 6.0 para la implementación de los algoritmos, mientras que la parte visual será desarrollada en Visual Basic 6.0.

El sistema operativo sobre el cuál se trabajará será Windows 95 y/o Windows 98 2da. edición, por ser las plataformas disponibles para poder programar en Visual C++ o Visual Basic.

Los algoritmos serán implementados como dll's que serán llamadas desde Visual Basic ya que este último provee de las herramientas para crear una interface gráfica amigable.

3. Estructuras de datos

Se decidió utilizar lista simplemente enlazada y ordenada para la implementación de la clase Lista Genérica, debido a que optimiza el uso de la memoria y su sencillez permite un corto tiempo de desarrollo y testeo [10].

Para la clase Heap, se decidió usar una implementación con arrays [10]. Donde el hijo izquierdo del nodo del lugar i se encuentra en el lugar 2*i y el derecho en el 2*i+1 (suponiendo que el primer nodo se encuentra en el lugar 1 del array).

4. Generalidades

Se plantean aquellas generalidades del diseño que es necesario conocer y tener en cuenta durante las pruebas de los distintos algoritmos:

Configuración Regional
En cuanto a los seteos regionales: se deberá configurar en el panel de control el punto (.) como separador decimal y la coma (,) como separador de miles. Esto es importante pues de lo contrario, al momento de realizar la lectura o escritura de los archivos que contienen los datos, ocurrirá que el formato no será el correcto.

Dimensiones
Se utilizan valores normalizados, es decir entre 0 y 1, para las coordenadas x e y , pero además para la ventana de tiempo t.
De esta forma se resuelve el problema de escala que pueda existir al momento de utilizar estos valores, de forma que los resultados no se vean afectados por este problema.

Rock
$f(\theta)$ no se estima sino que es un parámetro. En la Sección IV se plantea una forma de estimar este valor aunque en nuestro caso, y teniendo en cuenta la discusión realizada en la Sección IV, parte 2.2.2 Estimación de $f(\theta)$, $f(\theta)$ ha de ser un parámetro que el usuario ingresará. Esta decisión es

conveniente debido que disminuye el costo de aplicar el algoritmo Rock y sabemos que un valor aproximado aunque inexacto de f(θ) no tiene gran impacto sobre el resultado final.

Criterios de distancias entre puntos y clusters

Para el algoritmo de Aglomeración, de todos los criterios detallados en la Sección II Análisis de Requerimientos, se ha de implementar los siguientes:

	Suma Ponderada	Distancia de cuadra	Distancia de Chebychev	Angulo
Single linkage	SI	SI	SI	SI
Complete linkage	Si	SI	SI	SI
UPGMC	SI	SI	SI	SI

Para el algoritmo Rock, se utilizan los cuatro criterios de distancia entre puntos que se han de utilizar para Aglomeración.

VII VERIFICACIÓN

1. Objetivo

El objetivo de esta sección es documentar la verificación de los diferentes módulos desarrollados. Las pruebas que se documentan aquí son el resultado del testeo final a partir del cual los módulos y algoritmos fueron aceptados. Se presentan las observaciones obtenidas luego del último test de cada uno de los módulos.

2. Testeo de módulos

A continuación se resumen las pruebas realizadas a los módulos descriptos en la Sección V Diseño de módulos.

Módulo Lista genérica

Se realizaron pruebas instanciando la lista con elementos de tipo: Entero. Se probaron todas las operaciones de la clase en secuencia de forma tal de chequear tanto los casos intermedios como los casos límite (lista con un elemento, lista vacía). Búsqueda de elementos pertenecientes a la lista y de elementos que no pertenecen a ella. Las pruebas se realizaron paso a paso y visualizando un listado de los elementos de la lista luego de cada operación. También se testearon los constructores, operadores y destructor. Los resultados de las pruebas fueron satisfactorios por lo que el módulo Lista Genérica se considera aceptable.

Módulo heap genérico

Se realizaron pruebas instanciado el heap con elementos de tipo: Entero. Se probaron todas las operaciones de la clase en secuencia de forma tal de chequear tanto los casos intermedios como los casos límite (heap con un elemento, heap vacío, heap lleno). Búsqueda de elementos pertenecientes al heap y de elementos de que no pertenecen el. Las pruebas se realizaron paso a paso y visualizando un listado por nivel de los nodos del heap luego de cada operación. También se testearon los constructores, operadores y destructor. Los resultados de las pruebas fueron satisfactorios por lo que el módulo Heap Genérico se considera aceptable.

Módulo punto_dato

Se realizaron pruebas creando y asignando las propiedades de un objeto de esta clase, luego visualizando los datos. También modificando los datos de un punto_dato previamente creado. También se testearon los constructores, operadores y destructor. Los resultados de las pruebas fueron satisfactorios por lo que el módulo Punto_Dato se considera aceptable.

Módulo Cluster

Se probaron todas las operaciones de la clase en secuencia de forma tal de chequear tanto los casos intermedios como los casos límite (cluster con un elemento, heap vacío, heap lleno). Principalmente la operación Combinar clusters y dar lista de puntos y lista no compatibles. Las pruebas se realizaron paso a paso y visualizando un listado de los puntos del cluster luego de cada operación. También se testearon los constructores, operadores y destructor. Los resultados de las pruebas fueron satisfactorios por lo que el módulo Cluster se considera aceptable.

Módulos Distancia, DistanciaPunto y DistanciaCluster

Estos tres módulos se testearon en conjunto ya que el módulo distancia simplemente redirecciona la llamada a la operación que corresponde según sea distancia entre puntos o clusters y según la función de distancia. La verificación se realizó calculando aparte (en forma manual) la distancia entre puntos y clusters y verificando que dieran igual. Por razones de tiempo, no todas las funciones de distancia entre clusters fueron utilizadas por lo tanto se ha omitido el testeo de aquellas que no fueron utilizadas y son las siguientes:

- CentrosPonderados
- PromParesNoPonderados
- PromParesPonderados

Para el resto de las funciones de distancia, los resultados de las pruebas fueron satisfactorios por lo que los módulos Distancia, DistanciaPunto y DistanciaCluster se consideran aceptables.

Módulo IOPuntosDato
Este módulo realiza el ingreso de los datos y arma el heap global inicial que se usa en el módulo Algoritmo. La prueba de este módulo es realizada utilizando los archivos de trazas generados en el módulo Algoritmo.

Módulo Algoritmos
La prueba de este algoritmo se realizó estudiando los archivos de trazas que opcionalmente deja el algoritmo.
Es posible generar los siguientes archivos:

- Histórico de clusters generados en cada paso del algoritmo. Permite el seguimiento de las combinaciones de clusters realizadas en cada paso. Se genera para AGLOMERACION y ROCK.
- Histórico de contenido del heap global. Permite conocer el estado de los heaps (global y locales) a partir de los cuales el algoritmo determina los clusters que debe combinar en cada momento. Se genera para AGLOMERACION y ROCK.
- Matriz de links. Solo es generado para algoritmo ROCK.

Con el archivo de histórico de clusters, es posible visualizar todos los clusters generados en cada paso del algoritmo. A partir de este archivo también es posible verificar que luego de la combinación de dos clusters se realizan correctamente las siguientes acciones: eliminación de los dos clusters que se combinan, inserción del cluster resultado de la combinación.

Con el archivo de histórico de contenido del heap global, es posible visualizar el estado del heap a partir del cual el algoritmo determina el par de clusters a combinar. Es posible verificar que se realizan correctamente las siguientes acciones: selección del par de clusters cuya distancia es mínima y luego de la combinación de dos clusters: eliminación de las referencias a los clusters combinados, inserción del nuevo cluster en el heap global y actualización del heap global y los heaps locales reflejando estos cambios.

Para el algoritmo ROCK se agregó un tercer archivo que permite visualizar la matriz de links inicial. Esta matriz es utilizada para armar el heap global inicial. Este archivo permite verificar que se calcularon correctamente los links entre todo par de puntos.

A partir del estudio detallado de estos tres archivos, es posible verificar los algoritmos AGLOMERACION y ROCK. Los resultados de las pruebas fueron satisfactorios por lo que el módulo Algoritmos se considera aceptable

VIII. EVALUACIÓN DE LOS RESULTADOS

1. Objetivos

Nuestro objetivo es realizar una evaluación de los clusters que se obtienen mediante la ejecución de los algoritmos Aglomeración y Rock.

Para evaluar los resultados se realizan pruebas comparando en términos de criterios de bondad de clusters y también en términos del costo total del ruteo. Este último se mide como la suma de los largos de las rutas.

La estrategia utilizada para la comparación de los resultados es, dado un caso de prueba seguimos los siguientes pasos

- Paso 1: aplicar el algoritmo de clustering con cada combinación de criterios de distancia, obteniendo para cada caso una asignación de puntos, la distancia intracluster, la distancia intercluster y el tiempo de ejecución.
- Paso 2: con cada asignación obtenida en el paso 1, aplicamos el algoritmo de ruteo, obteniendo el largo de ruta (suma de los largos de rutas).
- Paso 3: aplicar las heurísticas de asignación, obteniendo con cada heurística otras asignaciones de puntos y el tiempo de ejecución.
- Paso 4: con cada asignación obtenida en el paso 3, aplicamos el algoritmo de ruteo, obteniendo el largo de ruta (suma de los largos de rutas).
- Paso 5: comparamos el mejor largo de ruta obtenido en el paso 2 contra el mejor largo de ruta obtenido en el paso 4.

Se siguen estos pasos tanto para Aglomeración como para Rock.

Para el caso del Rock, no existe combinación de criterios de distancia (distancia entre puntos y distancia entre clusters). Sin embargo, requiere de dos parámetros: Umbral y F(theta). Según la disposición de los puntos en cada caso de prueba, se eligen el Umbral y el F(theta) mas conveniente.
Si los puntos están dispersos, entonces se debe aumentar el umbral, de lo contrario conviene disminuirlo. Recordar que el umbral determina si dos puntos son vecinos de acuerdo a lo siguiente:

Dados dos puntos P1 y P2
Si se cumple que Dist(P1, P2) < umbral entonces son vecinos

Recordar también que el F(theta) se utiliza para estimar la cantidad de vecinos en cada cluster. Para mas detalles sobre estos parámetros ver Sección IV Adaptación al problema.

Los criterios de bondad que se utilizan son la distancia intracluster e intercluster, definidas mas adelante en esta sección.
Tanto las heurísticas de asignación como el algoritmo de ruteo utilizado para la comparación de los resultados fueron proporcionados por el usuario: Libertad Tansini y se describen mas adelante en esta sección.

2. Criterios de bondad

Para medir la bondad de los clusters se utilizaron la distancia intracluster e intercluster. Estas distancias son criterios de propósito general para evaluar soluciones de clustering. Se considera que la mejor solución de clustering es aquella que minimiza la distancia intracluster y maximiza la distancia intercluster.

De esta forma, uno de los criterios de bondad utilizado es el promedio de distancias intracluster de todos los clusters de la solución (ver Anexo I – Estado del arte de Clustering). La fórmula es la siguiente:

$$\text{Promedio de distancias intracluster} = \frac{\sum_{i=1}^{i=N} \text{DistanciaIntraCluster}(Ci)}{N}$$

donde N es la cantidad de clusters de la solución.

Otro de los criterios de bondad utiliza las distancias intercluster. Esta se calcula como el promedio de las distancias entre todo par de clusters de la solución. La fórmula es la siguiente:

$$\text{Promedio de distancias intercluster} = \frac{\sum_{i=1}^{i=N-1} \sum_{j=i+1 \, j \neq i}^{j=N} \text{DistanciaInterCluster}(Ci, Cj)}{N(N-1)}$$

donde N es la cantidad de clusters de la solución.

Además de estos criterios de bondad típicos para evaluar soluciones de clustering, se utiliza un tercer criterio que mide la bondad de los clusters en términos de costo de ruteo. Esto tiene sentido, puesto que el objetivo es aplicar algoritmos de clustering para resolver la fase de asignación de una solución de MDVRPTW basada en un enfoque de Asignación y Ruteo. El costo de ruteo se calcula como la suma de los largos de rutas de la solución de ruteo.

$$\text{Costo de ruteo} = \sum_{i=1}^{R} \text{LargoRuta}(ri)$$

donde R es la cantidad de rutas obtenidas por el algoritmo de ruteo utilizado.

2.1 Distancia intracluster

La distancia intracluster es una función que permite estimar el grado de similitud entre los puntos de un mismo cluster. Uno de los objetivos de los algoritmos de clustering es minimizar esta distancia.
Para medir esta distancia tenemos dos alternativas (ver Anexo I Estado del arte del clustering):

Alternativa 1
La distancia intracluster es la varianza, o sea la sumatoria de los cuadrados de las distancias de cada punto del cluster con la media o esperanza. La media o esperanza de un cluster es el centroide.

Alternativa 2
La distancia intracluster es el promedio de las distancias entre todo par de puntos de un cluster.

Seleccionamos la alternativa 1 por las siguientes razones:
- Menor costo de cálculo.
- Menor sensibilidad a los outliers. La alternativa 2 es altamente sensible a los outliers debido a que realiza el cálculo en base a promedios.

La distancia intracluster de un cluster Ci se calcula de la siguiente manera:

$$\text{DistanciaIntraCluster}(Ci) = \sum_{pi \in Ci} \text{distancia}(\text{SumaPonderada},1,1, pi,c)^{2}$$

donde c es el centroide del cluster Ci y n es la cantidad de puntos en el cluster Ci.

La distancia intracluster requiere de una función de distancia entre puntos. En el análisis de requerimientos se definieron cuatro funciones de distancia entre puntos. Para que tenga sentido la comparación entre diferentes resultados de los algoritmos de clustering, independientemente de la función de distancia utilizada, la función de distancia entre puntos para evaluar el resultado de clustering debe ser siempre la misma. En este sentido, alcanza con elegir cualquiera de las cuatro distancias entre puntos definidas, ya que lo único que tiene valor es la comparación, no el valor en si mismo. Por esta razón, en todos los casos, la distancia intracluster se calcula en base a la distancia

entre puntos que llamamos SUMA PONDERADA. El segundo y tercer parámetro de la función de distancia corresponden a wxy y wt, según se definen en el Análisis de Requerimientos.

2.2 Distancia intercluster

La distancia intercluster es una función que permite estimar el grado de separación entre los distintos clusters. Uno de los objetivos de los algoritmos de clustering es maximizar esta distancia.
La distancia intercluster requiere de una función de distancia entre clusters. Como se detalla en la Sección VI implementación, se implementaron tres funciones de distancia entre clusters. Para que tenga sentido la comparación entre diferentes resultados de los algoritmos de clustering, independientemente de la función de distancia utilizada, la función de distancia entre clusters para evaluar el resultado de clustering debe ser siempre la misma. En este sentido, alcanza con elegir cualquiera de las funciones de distancia entre clusters definidas, ya que lo único que tiene valor es la comparación, no el valor en si mismo. Por esta razón, en todos los casos, la distancia intercluster se calcula en base a la distancia entre clusters que llamamos UPGMC.

2.3 Largo de Ruta

Una vez que ejecutamos uno de los algoritmos de clustering obtenemos una asignación de clientes a depósitos. Con esta asignación, es posible aplicar el algoritmo de ruteo y de esta forma obtener el largo total de las rutas. Esta medida también es una medida de bondad de clusters. Cuanto menor sea el largo total de las rutas, indirectamente, podemos decir que mejor es la solución de clustering.
El algoritmo de ruteo utilizado, en este trabajo, fue proporcionado por el usuario: Libertad Tansini.

3. Resultados

Todas las pruebas han sido realizadas en una PC Pentium III con 64 MB de Ram.

Para cada algoritmo y caso de prueba, se presenta una tabla detallando la distancia intracluster, la distancia intercluster, el tiempo de ejecución del algoritmo, medido en segundos, y el largo total de la ruta obtenida mediante el algoritmo de ruteo.

Los parámetros para el algoritmo de ruteo son en todos los casos los mismos:
- capacidad del vehículo = 250 unidades
- tiempo de espera máxima = 4:30 horas
- velocidad del vehículo = 90 unidades
- constante escalar de distancia = 0,01

No se especifica las unidades de medida de la capacidad y velocidad del vehículo, pues se asumen unidades ficticias. Esto se debe a que los datos de Laporte [55] utilizados para las pruebas no especificaban unidades.
En el caso del tiempo, las unidades de medida son horas y minutos. De aquí en mas, se tomarán como unidades de medida las antes mencionadas, cuando se hable de tiempo de espera, capacidad y velocidad del vehículo.

En el caso del algoritmo de Aglomeración se utilizan dos tipos de funciones de distancia: una que mide la distancia entre puntos y otra función que mide la distancia entre clusters. De la combinación de estas funciones de distancia surgen los distintos resultados de clustering para un mismo caso. Las columnas de las tablas, representan las funciones de distancia entre puntos, mientras que las filas, representan las funciones de distancia entre clusters. Para cada combinación se especifican los resultados obtenidos, incluyendo el promedio de distancia intracluster, el promedio de distancia intercluster, el tiempo de ejecución y el largo de ruta.

En el caso de Rock, no se utiliza distancia entre clusters, pero el algoritmo recibe dos parámetros: F(theta) y Umbral. Estos se eligieron en base a pruebas de sensibilidad de forma de determinar aquellos valores para los cuales la distancia total de las rutas sean mínimas y además el número de clientes que queden sin asignar también sea mínimo. En la tabla se incluyen, para cada función de distancia entre puntos, los valores de distancia intracluster, intercluster, tiempo de ejecución y largo de ruta obtenidos para un valor fijo de los parámetros F(theta) y Umbral.

Luego de cada tabla se presenta un resumen de las características del caso en el que se obtuvo el menor largo de ruta (en la tabla este valor se resalta en color rojo). En base a esto, se busca determinar para cada caso, cual fue el algoritmo y cuales son las combinaciones de distancia que arrojaron mejores resultados.

Por otro lado, se disponen de cuatro heurísticas que realizan asignación de clientes a depósitos y no se basan en técnicas de clustering [13]. Hallando una asignación con estas heurísticas y una asignación con los algoritmos de clustering es posible rutear sobre ambas asignaciones y luego comparar cual obtuvo el menor largo de ruta.
Básicamente, los casos que se han utilizado para las pruebas corresponden al reporte técnico CRT-95-76 (Cordeau, Gendreau and Laporte) [55]. Excepto los casos 5 y 6 que fueron creados para las pruebas y que representan un conjunto de clientes que se encuentran distribuidos de forma simétrica con respecto a los depósitos. En este caso resulta mas fácil analizar visualmente el comportamiento de los algoritmos de clustering.

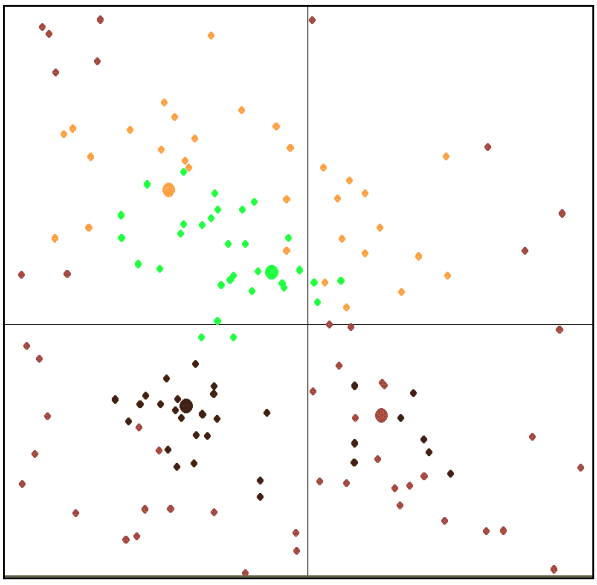
Los casos seleccionados para las pruebas, contemplan tanto problemas de tamaño pequeño, como mediano y grande, de acuerdo a la cantidad de clientes y depósitos. Es importante aclarar, que no se conocen para estos casos de prueba la mejor solución de ruteo que contemple todas las restricciones impuestas a nuestro problema original. En particular los casos de Laporte [55] fueron concebidos para un problema de ruteo pero con otras restricciones no comparables con las nuestras.

En la visualización de cada caso, se representan mediante puntos los clientes y los depósitos. Los clientes en color rosa y los depósitos en un tamaño mayor y color azul.

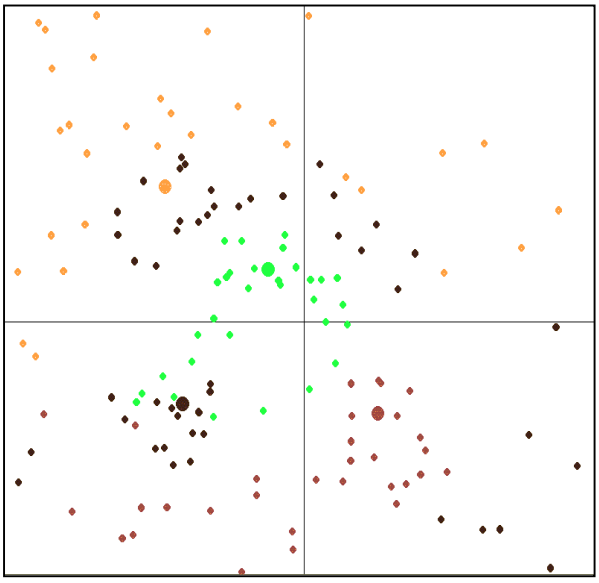
4. Cómo varia la forma de un cluster

En esta sección se busca mostrar mediante imágenes un ejemplo de como varia la forma de un cluster, según las funciones de distancia entre puntos que se utilicen.
A cada depósito se le ha asociado un color diferente. Cada cluster resultante se identifica con el color del depósito que contiene. De esta forma es muy fácil visualizar las diferencias de asignación cuando se cambia la función de distancia entre puntos. Es decir, todos los clientes y el depósito que pertenecen a un mismo cluster están dibujados siempre con el mismo color.

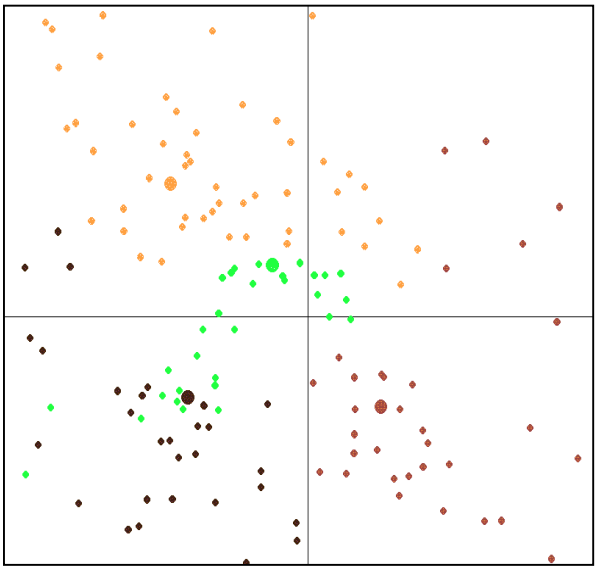
Se ha elegido el algoritmo Rock y el caso 7 de la sección anterior, como ejemplo para mostrar como varia la forma de los clusters según se utilizan las diferentes funciones de distancia.
Para ello se incluyen a continuación, cuatro imágenes donde se muestra la asignación de clientes que produjo el clustering. Para el resto de los casos es posible visualizar estas diferencias utilizando directamente la aplicación para pruebas de clustering desarrollado.



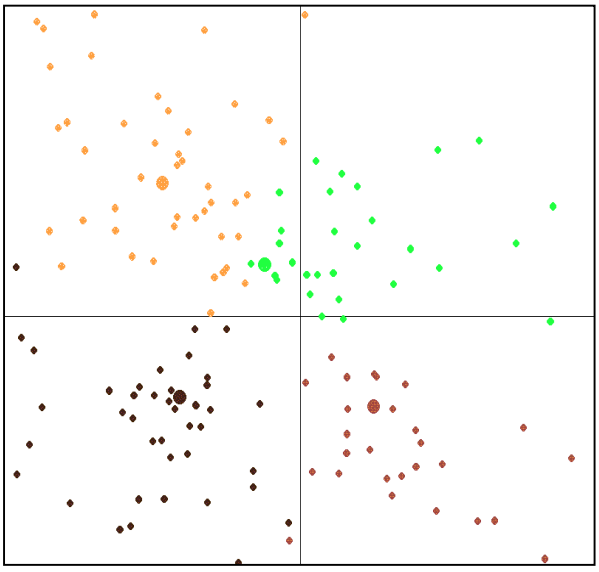
Suma ponderada. Largo de ruta = 58.8



Cuadra. Largo de ruta 46.33



Chebychev. Largo de ruta = 55.8



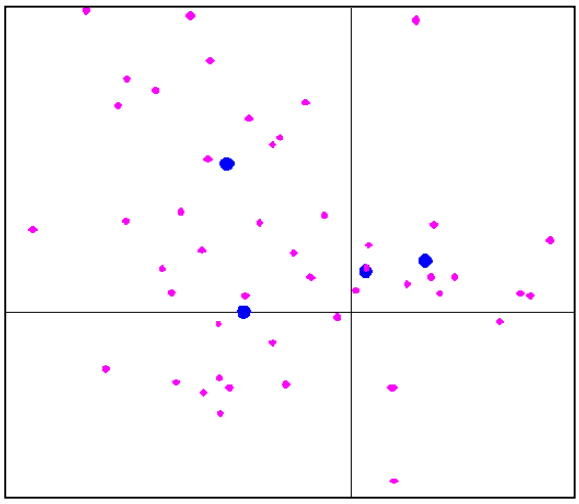
Angulo. Largo de ruta 30.08

Observar como el resultado utilizando distancia ángulo es intuitivamente el mejor clustering ya que los clusters se ven más compactos en su interior y más separados entre ellos. Los resultados de las pruebas, en términos de largo de ruta, confirman esta percepción.

4.1 Caso 1

LNC* vacía, archivo PR01.txt – Laporte
Nro. de Depósitos = 4
Nro. de Clientes = 48

En este caso, las lista de depósitos no compatibles es vacía para todos los clientes.
Gráficamente la distribución de estos depósitos y clientes es la siguiente:

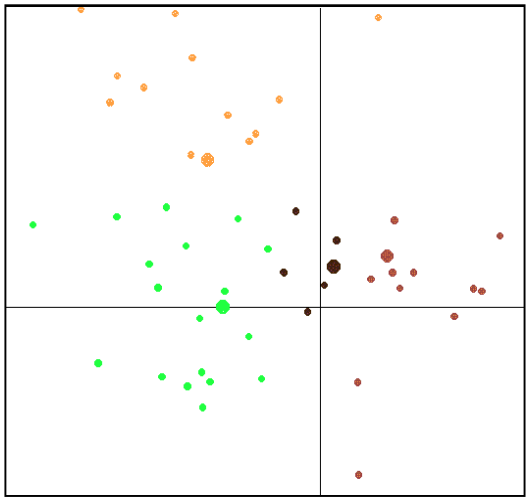


Resultados de heurísticas de asignación			
Heurística	**Tiempo de Ejecución**	**Clientes no asignados**	**Largo de Ruta**
1	< 1	12	17.7
2	< 1	12	17.7
3	< 1	12	19.84
5	< 1	12	17.7

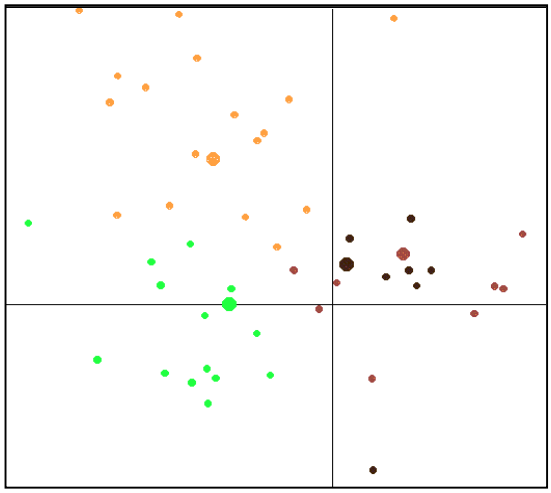
La heurística con la que se obtiene la mejor solución (menor largo de ruta) corresponde a:
- Heurística = 1,2 y 5
- Largo de ruta = 17,7
- Tiempo de ejecución <1

Las asignaciones con las que se obtuvo menor largo de ruta se visualizan en las dos figuras siguientes:

* LNC es la abreviación de Lista de No Compatibles



Aglomeración: Angulo y Upgmc



Rock: Angulo

Los datos obtenidos se resumen en la siguiente tabla.

Resultados de ROCK				
F(Theta) =0.1 Umbral = 0.6	**Suma Ponderada**	**Distancia de Cuadra**	**Distancia de Chebychev**	**Angulo**
Promedio intracluster	2.65	3.36	3.27	1.56
Promedio intercluster	0.75	0.41	0.42	0.82
Tiempo de Ejecución	< 1	< 1	< 1	< 1
Largo de Ruta	19.67	17.56	26.66	15.88

La función de distancia con la que se obtiene la mejor solución (menor largo de ruta) corresponde a:
- Distancia entre puntos = Angulo
- Largo de ruta = 15.88

De los criterios de bondad se observa:
- Rango en que varía el promedio de distancia intracluster = [1.56 .. 3.36]
- Rango en que varía el promedio de distancia intercluster = [0,41 .. 0,82]
- Distancia intracluster para menor largo de ruta = 1.56
- Distancia intercluster para menor largo de ruta = 0.82

Resultados de Aglomeración				
	Suma Ponderada	**Distancia de Cuadra**	**Distancia de Chebychev**	**Angulo**
Promedio intracluster				
Single linkage	1.53	1.43	1.37	1.23
Complete linkage	1.38	1.43	1.32	1.23
Upgmc	1.46	1.38	1.33	1.36
Promedio intercluster				
Single linkage	0.94	0.97	0.97	0.99
Complete linkage	0.92	0.91	0.95	0.99
Upgmc	1.02	0.94	0.95	0.93
Tiempo de Ejecución				
Single linkage	< 1	< 1	< 1	< 1
Complete linkage	< 1	< 1	< 1	< 1
Upgmc	< 1	< 1	< 1	< 1
Largo de Ruta				
Single linkage	16.84	18.2	14.99	15.69
Complete linkage	15.26	15.68	14.65	15.69
Upgmc	16.84	14.65	14.98	13.57

La combinación de criterios de distancia con la que se obtiene la mejor solución (menor largo de ruta) corresponde a:

- Largo = 13,57
- Distancia entre clusters = Upgmc
- Distancia entre puntos = Angulo

De los criterios de bondad se observa:

- Rango en que varía el promedio de distancia Intracluster = [1,23 .. 1,53]
- Rango en que varía el promedio de distancia Intercluster = [0,91 .. 1,02]
- Distancia Intracluster para menor largo de ruta = 1,36
- Distancia Intercluster para menor largo de ruta = 0,93

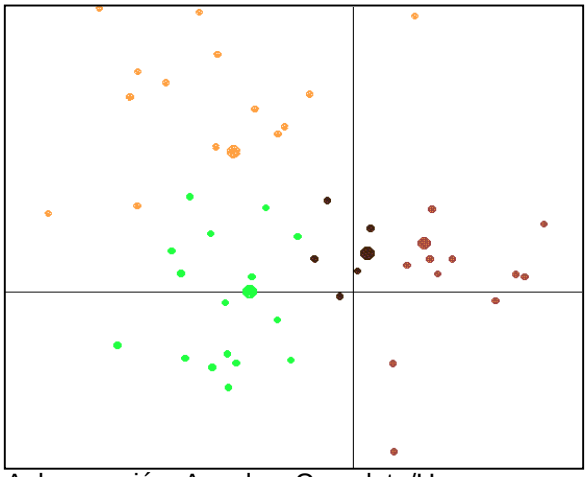
En este caso la mejor solución se obtuvo con la distancia ángulo tanto para el algoritmo de Aglomeración como para el algoritmo Rock. El menor largo de ruta se obtuvo con el algoritmo Aglomeración.

4.2 Caso 2

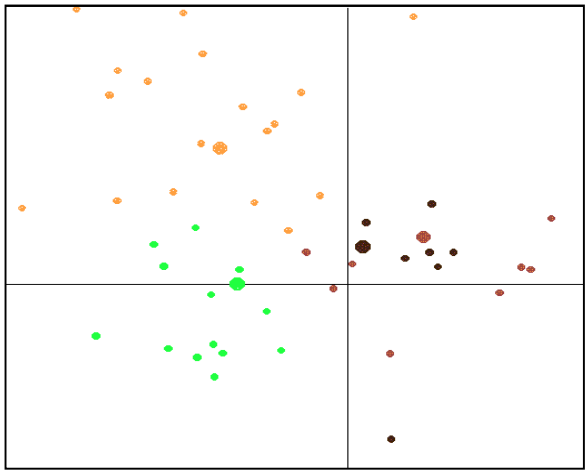
LNC* no vacía, archivo PR01.txt – Laporte
Nro. de Depósitos = 4
Nro. de Clientes = 48

En este caso existen restricciones de compatibilidad entre clientes y depósitos (LNC no vacía)
Gráficamente la distribución de estos depósitos y clientes es la misma que en el Caso 1

Las asignaciones con las que se obtuvo menor largo de ruta se visualizan en las dos figuras siguientes:



Aglomeración: Angulo y Complete/Upgmc



Rock: Angulo

Resultados de Rock				
F(Theta) =0.1 Umbral = 0.6	**Suma Ponderada**	**Distancia de Cuadra**	**Distancia de Chebychev**	**Angulo**
Promedio intracluster	2.15	2.6	2.83	1.54
Promedio intercluster	0.81	0.65	0.64	0.82
Tiempo de Ejecución	< 1	< 1	< 1	< 1
Largo de Ruta	17.14	18.74	18.62	15.88

* LNC es la abreviación de Lista de No Compatibles

Evaluación de los Resultados

La función de distancia con la que se obtiene la mejor solución (menor largo de ruta) corresponde a:
- Distancia entre puntos = Angulo
- Largo de ruta = 15.88

De los criterios de bondad se observa:
- Rango en que varía el promedio de distancia intracluster = [1.54 .. 2.83]
- Rango en que varía el promedio de distancia intercluster = [0,64 .. 0,82]
- Distancia intracluster para menor largo de ruta = 1.54
- Distancia intercluster para menor largo de ruta = 0.82

Resultados de Aglomeración				
	Suma Ponderada	**Distancia de Cuadra**	**Distancia de Chebychev**	**Angulo**
Promedio intracluster				
Single linkage	1.54	1.56	1.49	1.5
Complete linkage	1.49	1.49	1.42	1.45
Upgmc	1.4	1.39	1.41	1.41
Promedio intercluster				
Single linkage	0.93	0.92	0.88	0.87
Complete linkage	0.92	0.93	0.89	0.91
Upgmc	1.01	0.92	0.9	0.9
Tiempo de Ejecución				
Single linkage	< 1	< 1	< 1	< 1
Complete linkage	< 1	< 1	< 1	< 1
Upgmc	< 1	< 1	< 1	< 1
Largo de ruta				
Single linkage	16.84	16.76	15.02	15.01
Complete linkage	14.69	14.80	14.32	13.57
Upgmc	16.01	14.65	13.57	13.57

La combinación de criterios de distancia con la que se obtiene la mejor solución (menor largo de ruta) corresponde a:
- Distancia entre clusters = Complete / Upgmc
- Distancia entre puntos = Angulo / Chebychev
- Largo de ruta = 13.57

De los criterios de bondad se observa:
- Rango en que varía el promedio de distancia intracluster = [1.39 .. 1.56]
- Rango en que varía el promedio de distancia intercluster = [0,87 .. 1,01]
- Distancia intracluster para menor largo de ruta = 1,41 - 1,45
- Distancia intercluster para menor largo de ruta = 0.9 . 0.91

En este caso la mejor solución se obtuvo con la distancia ángulo tanto para el algoritmo de Aglomeración como para el algoritmo Rock. El menor largo de ruta se obtuvo con el algoritmo Aglomeración.

4.3 Caso 7

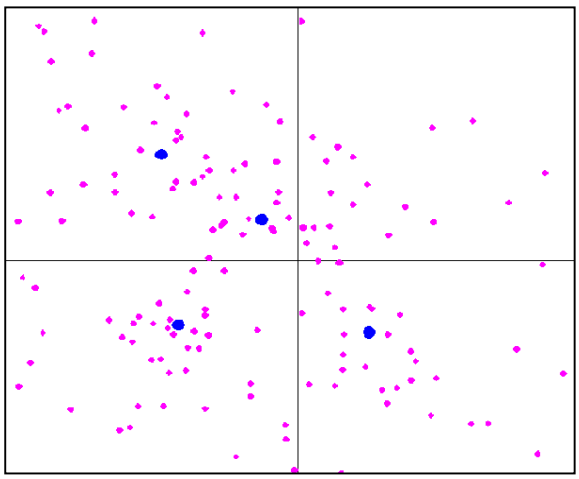
LNC* vacía, archivo PR03.txt - Laporte
Nro. de Depósitos = 4
Nro. de Clientes = 144

En este caso, la lista de depósitos no compatibles es vacía para todos los clientes.

* LNC es la abreviación de Lista de No Compatibles

Evaluación de los Resultados

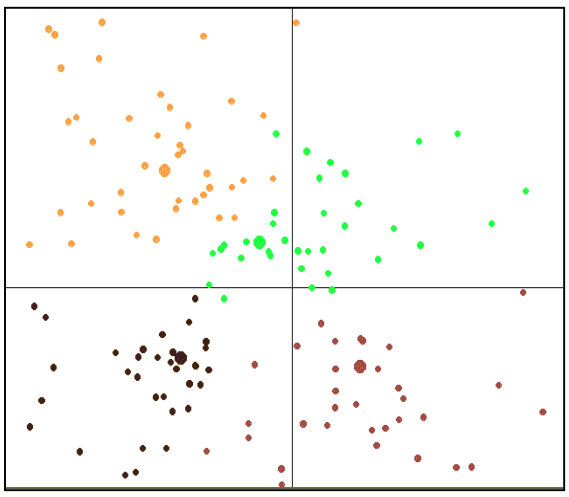
Gráficamente la distribución de estos depósitos y clientes es la siguiente:



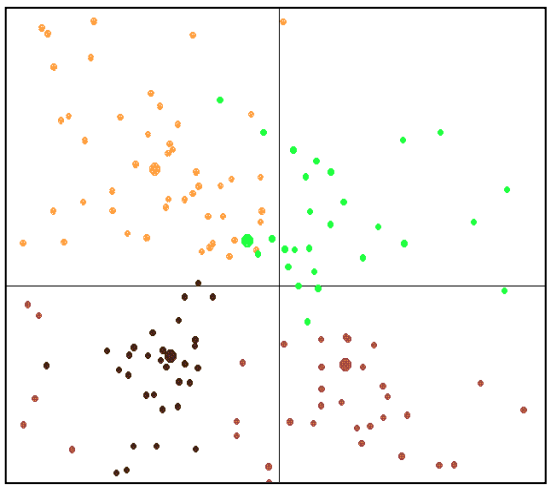
Resultados de heurísticas de asignación			
Heurística	**Tiempo de Ejecución**	**Clientes no asignados**	**Largo de Ruta**
1	< 1	0	68.97
2	< 1	0	68.97
3	1	0	70.64
5	< 1	0	69.35

La heurística con la que se obtiene la mejor solución (menor largo de ruta) corresponde a:
- Heurística = 1 y 2
- Largo de ruta = 68,97
- Tiempo de ejecución <1

Las asignaciones con las que se obtuvo menor largo de ruta se visualizan en las dos figuras siguientes:



Aglomeración: Suma y Upgmc



Rock: Angulo

Resultados de Rock				
F(Theta) =0.1 Umbral = 0.6	**Suma Ponderada**	**Distancia de Cuadra**	**Distancia de Chebychev**	**Angulo**
Promedio intracluster	11.17	10.88	11.08	5.61
Promedio intercluster	0.62	0.69	0.69	1.09
Tiempo de Ejecución	1	1	1	2
Largo de Ruta	58.8	46.33	55.8	30.08

La función de distancia con la que se obtiene la mejor solución (menor largo de ruta) corresponde a:
- Distancia entre puntos = Angulo
- Largo de ruta = 30.08

De los criterios de bondad se observa:
- Rango en que varía el promedio de distancia intracluster = [5.61 .. 11.17]
- Rango en que varía el promedio de distancia intercluster = [0,62 .. 1,09]
- Distancia intracluster para menor largo de ruta = 5.61
- Distancia intercluster para menor largo de ruta = 1.09

Resultados de Aglomeración				
	Suma Ponderada	**Distancia de Cuadra**	**Distancia de Chebychev**	**Angulo**
Promedio intracluster				
Single linkage	5.98	6.21	6.07	7.45
Complete linkage	5.99	6.16	4.59	6.46
Upgmc	4.52	4.88	4.63	4.53
Promedio intercluster				
Single linkage	1.14	1.25	1.12	0.92
Complete linkage	1.12	1.05	1.18	0.99
Upgmc	1.16	1.13	1.19	1.2
Tiempo de Ejecución				
Single linkage	< 1	< 1	< 1	1
Complete linkage	< 1	< 1	< 1	< 1
Upgmc	< 1	< 1	< 1	< 1
Largo de Ruta				
Single linkage	53.37	56.69	53.7	63.03
Complete linkage	36.31	31.35	31.32	62.13
Upgmc	29.1	32.13	31.92	30.48

La combinación de criterios de distancia con la que se obtiene la mejor solución (menor largo de ruta) corresponde a:
- Largo = 29.1
- Distancia entre clusters = Upgmc
- Distancia entre puntos = Suma ponderada

De los criterios de bondad se observa:
- Rango en que varía el promedio de distancia Intracluster = [4,52 .. 7,45]
- Rango en que varía el promedio de distancia Intercluster = [0,92 .. 1,25]
- Distancia Intracluster para menor largo de ruta = 4,52
- Distancia Intercluster para menor largo de ruta = 1,16

En este caso la mejor solución se obtuvo para el algoritmo de Aglomeración con la distancia suma ponderada y para el algoritmo Rock con la distancia ángulo. El menor largo de ruta se obtuvo con el algoritmo Aglomeración.

4.4 Caso 10

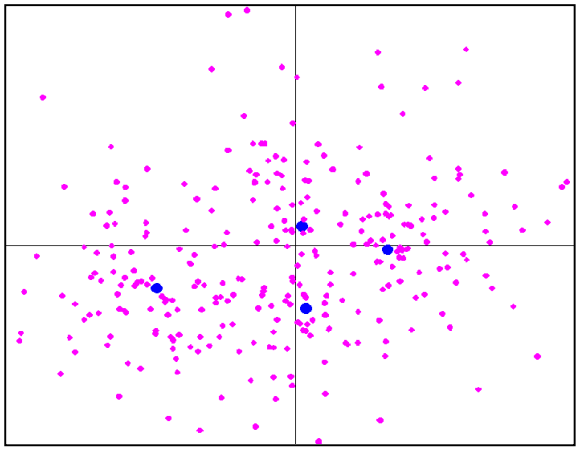
LNC* vacía, archivo PR06.txt - Laporte
Nro. de Depósitos = 4
Nro. de Clientes = 288

En este caso, las lista de depósitos no compatibles es vacía para todos los clientes.

* LNC es la abreviación de Lista de No Compatibles

Evaluación de los Resultados

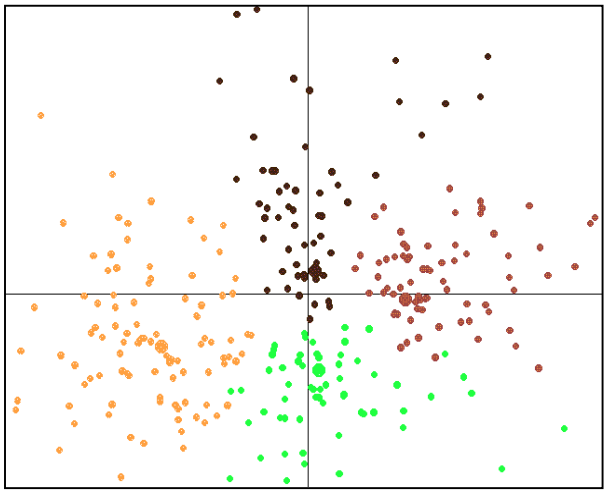
Gráficamente la distribución de estos depósitos y clientes es la siguiente:



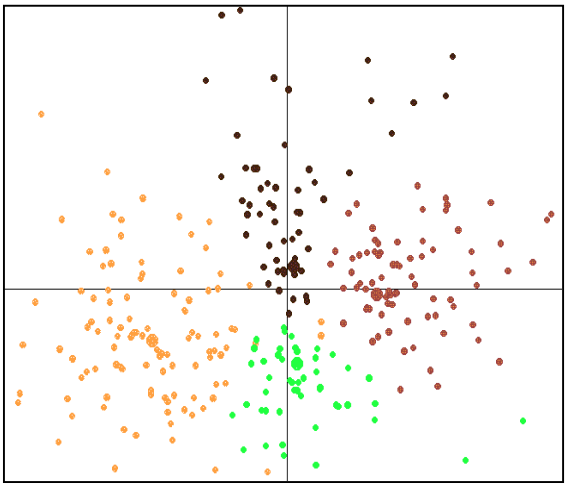
Resultados de heurísticas de asignación			
Heurística	**Tiempo de Ejecución**	**Clientes no asignados**	**Largo de Ruta**
1	< 1	0	93.7
2	< 1	0	93.7
3	< 1	0	157.81
5	< 1	0	93.7

La heurística con la que se obtiene la mejor solución (menor largo de ruta) corresponde a:
- Heurística = 1, 2 y 5
- Largo de ruta = 93,7
- Tiempo de ejecución <1

Las asignaciones con las que se obtuvo menor largo de ruta se visualizan en las dos figuras siguientes:



Aglomeración: Angulo y Single



Rock: Angulo

Resultados de Rock				
F(Theta) =0.1 Umbral = 0.3	**Suma Ponderada**	**Distancia de Cuadra**	**Distancia de Chebychev**	**Angulo**
Promedio intracluster	9.9	14.52	10.33	6.82
Promedio intercluster	0.78	0.52	0.73	0.92
Tiempo de Ejecución	18	19	18	4
Largo de Ruta	173.04	190.17	138.85	117.15

Evaluación de los Resultados

La función de distancia con la que se obtiene la mejor solución (menor largo de ruta) corresponde a:
- Distancia entre puntos = Angulo
- Largo de ruta = 117.15

De los criterios de bondad se observa:
- Rango en que varía el promedio de distancia intracluster = [6.82 .. 14.52]
- Rango en que varía el promedio de distancia intercluster = [0,52 .. 0.92]
- Distancia intracluster para menor largo de ruta = 6.82
- Distancia intercluster para menor largo de ruta = 0.92

Resultados Aglomeración				
	Suma Ponderada	**Distancia de Cuadra**	**Distancia de Chebychev**	**Angulo**
Promedio intracluster				
Single linkage	10.12	10.98	10.93	6.62
Complete linkage	8.92	7.52	7.05	6.71
Upgmc	6.58	6.75	6.68	6.57
Promedio intercluster				
Single linkage	0.77	0.81	0.84	0.92
Complete linkage	0.86	0.91	0.92	0.94
Upgmc	0.92	0.92	0.92	0.93
Tiempo de Ejecución				
Single linkage	3	2	3	4
Complete linkage	3	2	2	4
Upgmc	2	2	2	3
Largo de Ruta				
Single linkage	138.38	156.06	166.96	110.09
Complete linkage	153.97	125.39	131.53	124.39
Upgmc	116.24	114.1	117.21	114.2

La combinación de criterios de distancia con la que se obtiene la mejor solución (menor largo de ruta) corresponde a:
- Largo de ruta = 110.09
- Distancia entre clusters = Single linkage
- Distancia entre puntos = Angulo

De los criterios de bondad se observa:
- Rango en que varía el promedio de distancia Intracluster = [6,57 .. 10,98]
- Rango en que varía el promedio de distancia Intercluster = [0,77 .. 0,94]
- Distancia Intracluster para menor largo de ruta = 6.62
- Distancia Intercluster para menor largo de ruta = 0,92

En este caso la mejor solución se obtuvo con la distancia ángulo tanto para el algoritmo de Aglomeración como para el algoritmo Rock. El menor largo de ruta se obtuvo con las heurísticas de asignación 1, 2 y 5.

5 Conclusiones de los resultados obtenidos

Se presenta un análisis de los resultados obtenidos, a partir de las pruebas realizadas, utilizando los casos que se incluyen anteriormente más los casos del Anexo III. En particular, los casos considerados para el cálculo de los porcentajes que se muestran en los gráficos siguientes, son aquellos que no tienen restricciones de incompatibilidad entre clientes y depósitos. Esto es necesario, puesto que las heurísticas de asignación no tienen en cuenta tales restricciones.
En base a este análisis, evaluamos la aplicabilidad del clustering jerárquico, en particular los algoritmos AGLOMERACION y ROCK, al MDVRPTW.

Algoritmos
En la mayoría de los casos el algoritmo que obtuvo mejores resultados fue Aglomeración.

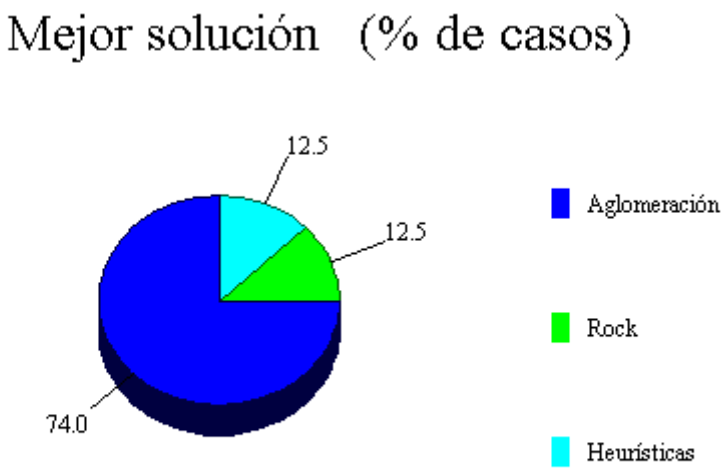


Gráfico 1

Recordar que nos interesa determinar con cual de los algoritmos (Aglomeración, Rock o Heurísticas) se obtiene una asignación a partir de la cual el ruteo obtiene la ruta de menor costo. Es en este sentido que diremos que un algoritmo tuvo mejor comportamiento con respecto a otro si a partir de la asignación que devuelve se obtiene una ruta de menor costo.

A partir del gráfico 1, claramente se observa que el comportamiento del algoritmo Aglomeración fue mejor que Rock e incluso que las heurísticas. Con respecto al algoritmo Rock, era de esperarse esta diferencia puesto que fue originalmente diseñado para datos categóricos. En este taller se ha realizado una adaptación de Rock para ajustarse a datos no categóricos y al resto de los requerimientos. Por más detalles ver la Sección IV Adaptación al problema. El algoritmo de Aglomeración es un algoritmo de propósito general y por ende era de esperar que se comportara mejor que Rock. En cuanto a las heurísticas, es interesante notar las mejoras obtenidas con el clustering puesto que, hasta el momento, estas heurísticas son de los pocos algoritmos desarrollados específicamente para resolver este problema. También es necesario tener en cuenta que los tiempos de ejecución de las heurísticas son menores que Aglomeración.

Según los resultados obtenidos, es posible clasificar la mejor opción según la cantidad de puntos del caso. Para casos en que la cantidad total de puntos es menor que 150 conviene utilizar Aglomeración como algoritmo de asignación. Para casos con más puntos conviene utilizar alguna de las heurísticas.

Funciones de distancia entre puntos
Tanto para el algoritmo de Aglomeración como para Rock, la distancia entre puntos con la cual se obtuvo en la mayoría de los casos las mejores soluciones es la distancia Angulo. En el gráfico siguiente es posible observar esto.

AGLOMERACIÓN

ROCK

Gráfico 2

Uno de nuestros objetivos era la búsqueda y el testeo de funciones alternativas de distancia. La idea es incluir la ventana de tiempo como un valor más a tener en cuenta para hallar la distancia entre dos puntos. De esta forma, obtener mejores resultados de clustering que ayuden a alcanzar mejores resultados de ruteo.
Es interesante notar, cómo a partir de las pruebas con varias funciones para medir distancia entre puntos, se descubrió que una de las funciones que a priori no parecía la mejor de las candidatas, luego resultó ser aquella con la cual se obtuvo mejores resultados. Esta función de distancia es la que llamamos Angulo. Queda en evidencia la importancia de las pruebas realizadas ya que a priori no era evidente este resultado, sobre todo para el algoritmo Rock con el cuál en un 90% de los casos obtuvo la asignación a partir de la cual se alcanza menor largo de ruta.

Otra de las funciones de distancia entre puntos con la que se obtuvo buenos resultados fue la Suma Ponderada. A priori, ésta era la función de distancia candidata a obtener mejores resultados puesto que está basada en la distancia euclídea y es la más comunmente usada en algoritmos de clustering.
Recordar que se calcula como la distancia euclídea más la diferencia entre los centros de la ventana de tiempo.

De los gráficos 1 y 2, se observa que la mejor combinación de Algoritmo y función de distancia entre puntos es Aglomeración con distancia Angulo.

Funciones de distancia entre clusters
La distancia entre clusters se usa solamente en el algoritmo de Aglomeración. Como se observa en el gráfico 3, en la mayoría de los casos las mejores soluciones se obtuvieron con la distancia upgmc.

Gráfico 3

Este resultado era de esperarse puesto que la distancia Upgmc es la distancia entre los centros de los clusters y que Single y Complete (distancia entre los puntos mas cercanos y alejados respectivamente) son altamente sensibles a los outliers. Recordar que los outliers son puntos inusualmente alejados del resto.

Sobre los criterios de bondad

Uno de nuestros objetivos es observar si existe una relación entre la medida de bondad de clusters y el largo de ruta obtenido por el ruteo, de forma tal de poder afirmar que cuanto mejor es una solución de clustering de acuerdo a los criterios de bondad se obtiene menor costo de ruteo.

Los criterios de bondad utilizados son la distancia intra e intercluster. Con respecto a la distancia intercluster no se encontró relación alguna. Esto es razonable, puesto que esta distancia mide cuan separados están los clusters entre ellos y es de esperar que el resultado del ruteo no sea afectado por esto.

En cuanto a la distancia intracluster, mide cuan cercanos están los puntos dentro de un cluster, es razonable pensar que a menor distancia intracluster se obtiene menor largo de ruta. Observamos que para el algoritmo Aglomeración la tendencia indica una relación creciente entre la distancia intracluster y largo de ruta, sin embargo, esta tendencia no está muy marcada. En cambio para el algoritmo Rock se observa claramente que a menor distancia intracluster se obtiene menor largo de ruta.

En resumen

Del análisis realizado anteriormente creemos que vale la pena investigar algoritmos de clustering para aplicarlos al problema MDVRPTW. En cuanto a la inclusión de la ventana de tiempo en los algoritmos de clustering se probaron varias formas de incluir este dato en el cálculo de distancia entre puntos. La distancia ángulo fue la mas novedosa y a la vez la que dio los mejores resultados, tanto en lo que tiene que ver con criterios de bondad típicos de clustering como en cuanto a los resultados posteriores en el ruteo.

IX. TRABAJO FUTURO

Asignar y luego rutear aparece como la alternativa indicada para problemas de tamaño considerable en MDVRPTW. Sin embargo, para la resolución de la fase de asignación se cuenta con unas pocas técnicas típicas, más las desarrolladas en New Assignment algorithms for the Multi-Depot Vehicle Routing Problem [13]. Es por esto que resulta novedoso aplicar técnicas de clustering para la resolución de la fase de asignación. Este trabajo considera la aplicación de una de las técnicas más típicas de clustering, el clustering jerárquico, si bien a la hora de ajustarse a los requerimientos particulares del problema se encuentran más puntos en contra que a favor para su aplicación (Ver Sección IV Adaptación al problema), esto conduce a pensar que el clustering jerárquico no es el más indicado en este caso, y que vale la pena continuar con el estudio de otras técnicas y algoritmos de clustering para su aplicación a la fase de asignación. Sin embargo, los resultados obtenidos al comparar el clustering contra las heurísticas de asignación [13] han demostrado que, en general, se obtienen mejores resultados con el clustering, aunque las diferencias del costo de ruteo no son muy marcadas. Además, para casos en que la cantidad de clientes es del orden de 200 o más, los tiempos de ejecución de las heurísticas son sensiblemente menores que los tiempos de ejecución del clustering. Por lo tanto, si consideramos que los tiempos de ejecución son críticos para la aplicación, en muchos casos las mejoras que se obtienen en el costo total de ruteo no se justifican.

En la sección IV se presentan distintas alternativas para la aplicación de clustering jerárquico, una de ellas plantea liberar al problema de las restricciones de incompatibilidad entre clientes y depósitos, para obtener mejores resultados. Esta alternativa, plantea tener categorías de depósitos y aplicar clustering a cada categoría por separado. Este sería un buen punto de partida si se desea continuar con el estudio de clustering jerárquico aplicado a este problema.

En este estudio se incluye la ventana de tiempo utilizando el valor del centro de la ventana como una medida más durante el cálculo de las distancias y sin tener en cuenta el horario entero de servicio. Otro de los lineamientos a seguir sería desarrollar nuevas formas de incluir la ventana de tiempo durante el clustering.

Trabajar con distancias reales y no con distancias basadas en la euclídea también es una prueba factible de ser incluida y con poco esfuerzo.

Otra posible mejora a este trabajo, sería mayor dedicación en la interface para su integración con otros sistemas como por ejemplo Sistemas de Información Geográfica (SIG).

RESUMEN DEL PROYECTO

El objetivo de esta Sección es presentar un resumen del trabajo realizado para este taller.
Este taller surge de la inquietud de investigar la aplicación de algoritmos de clustering a la fase de asignación del problema de ruteo. Más precisamente, la propuesta de este trabajo surge a raíz de la tesis de Maestría de Libertad Tansini. Esta tesis estudia algoritmos de asignación para el MDVRPTW.

El resultado de este taller incluye:

- Estado del Arte del Clustering. Un informe completo de las técnicas y algoritmos de clustering actualmente conocidos.
- Informe completo sobre software que realiza clustering. Resumen de información técnica de varios productos comerciales. También se incluye información valiosa para el usuario y direcciones de internet donde obtener más información de cada producto.
- Adaptación de algoritmos de clustering para su aplicación al problema MDVRPTW. Estudio de dos algoritmos de clustering jerárquico, AGLOMERACION y ROCK, y adaptación de ellos para estudiar su aplicabilidad al problema.
- Desarrollo de estos dos algoritmos de clustering adaptados al problema. Análisis, diseño e implementación utilizando un enfoque orientado a objetos.
- Análisis de resultados. Evaluación de la aplicabilidad de los algoritmos al MDVRPTW.
- Desarrollo de una herramienta para el testeo, visualización y análisis de los resultados.

Por todo esto, consideramos que se han cumplido los objetivos de este taller. Más aún si tenemos en cuenta que, a partir del análisis de los resultados (Ver Sección VIII, punto 5 Conclusiones de los resultados obtenidos), podemos afirmar que vale la pena investigar algoritmos de clustering para aplicarlos al problema MDVRPTW.

BIBLIOGRAFÍA

Clustering

[1] Data Mining Techniques: For Marketing, Sales, and Customer Support.
Michael J. A. Berry, Gordon Linoff. 1era edición (27/05/1997) John Wiley & Sons; ISBN: 0471179809

[2] Lance, G. N. And Williams, W. T. 1967. A general theory of classificatory sorting strategies.
Computer Journal.

[3] S. Guha, R. Rastogi and K. Shim. CURE: An efficient algorithm for clustering large data bases.

[4] V. Ganti, J. Gehrke, R. Ramakrishnan. CACTUS: Clustering Categorical data using summaries.

[5] Eui-Hong Han, George Karypis and Vipin Kumar. CHAMELEON: Hierarchical clustering using dynamic modeling.

[6] S. Guha, R. Rastogi and K. Shim. ROCK: A robust clustering algorithm for categorical attributes.

[7] Anthoni K. H. Tung, Jean Hou, Jiawei Han. COE: Custering with obstacles entities.

[8] Eui-Hong Han, George Karypis and Vipin Kumar . Clustering in a High dimensional Space Using Hypergraph Models.

[9] Eui-Hong Han, George Karypis and Vipin Kumar. Multilevel Refinement for Hierarchical Clustering.

Programación

[10] Alfred V. AHO, John E. Hopcroft, Jeffrey D. Ullman. Data structures and algorithms.
Ed. Adison Wesley. ISBN 0-201-00023-7

[11] Fco. Javier Ceballos. Curso de Programación en C++: Programación orientada a objetos.
Ed. Addison-Wesley Iberoamericana. Edición RA-MA 1991 ISBN 0-201-62500-8

[12] J. Harrington, M. Spenik, H. Brumbaugh, C. Diamond. Visual Basic 5: Interactive Course.
Ed. Waite Group Press, 1997 – ISBN 1-57169-077-8

MDVRP

[13] Ing. Daniel Giosa, Ing. Libertad Tansini, Ing. Omar Viera, MSc. New Assignment algorithms for the Multi-Depot Vehicle Routing Problem. Dpto. Investigación Operativa, Instituto de Computación, Facultad
de Ingeniería, UDELAR.

[14] L. Bodin, B. Golden, A. Assad and M. Ball. Routing and scheduling of vehicles and crews: The state of the art. Computers and Operations Research Vol. 10 (1983)

Otros

[15] J. Rey Pastor, L.A. Santaló, M. Balanzat. Geometría Analítica.
Ed. Kapelusz 4° edición – Setiembre 1959

[16] C. Ghezzi, M. Jazayeri and D. Mandrioli. Fundamentals of Sotfware Engineering.
Ed. Prentice Hall 1991 ISBN 0-13-820432-2

DIRECCIONES EN INTERNET

Clustering

[17] www.statsoftinc.com/textbooks/stcluan.html Funciones de distancia entre puntos
(Ultimo acceso: 06/05/2000)

[18] www.web.cs.ualberta.ca/~oleg/quantization.html Aplicación de clustering a la cuantización de imágenes
(Ultimo acceso: 24/04/2000)

[19] www.carol.wins.uva.nl/~niels/Pub/cluster/cluster.html Clustering de líneas (imágenes)
Nota: Esta página no se pudo acceder correctamente. No se pudo visualizar las fórmulas.
Esta página fue encontrada el 27/4/2000 y se intentó acceder nuevamente el 24/5/2000 encontrandose el mismo problema.

[20] www.cs.wisc.edu/~olvi/olvi.html Mining and Monitoring Evolving Data
(Ultimo acceso: 12/05/2000)

[21] http://149.170.199.144/multivar/ca_alg.htm Joining Clusters
(Ultimo acceso:16/05/2000)

[22] www.almaden.ibm.com\cs\quest\publications.html.Automatic Subspace Clustering of High Dimensional Data for Data Minnig Aplications. (Ultimo acceso: 16/05/2000)

Software de Clustering
[23] Accrue Decision Series: http://www.accrue.com/products/decision.html
[24] Advisor Toolkit: http://www.csihq.com/
[25] Affinium: http://www.unicacorp.com/products/model.htm
[26] Alice d'isoft: www.alice-soft.com/products/index.html
[27[Clementine: http://www.spss.com/software/clementine
[28] Clustan Graphics 4: www.clustan.com
[29] Darwin: http://www.think.com/html/products/dartechdatasht.htm#one
[30] Data Detective: http://www.smr.nl/
[31] Data Mining Suite: http://www.dataming.com/dmsuite.htm
[32] DataMiner 3D: http://www.dimension5.sk/products/products.htm
[33] DataMiner Software Kit: http://www.data-miner.com
[34] DataMite: http://www.lpa.co.uk/dtm.html
[35] DataScope: http://www.cygron.hu
[36] DecisionWorks: http://www.asacorp.com/product/index.html
[37] DeltaMiner: http://194.152.41.50/english/Delta/Products/Miner/delta_miner.htm
[38] IBM Intelligent Miner for Data: http://www.software.ibm.com/data/iminer/fordata/index.html
[39] Knowledge Studio: http://www.angoss.com/kstudio/studio.htm
[40] MineSet Suite: http://www.sgi.com/software/mineset/
[41] Nuggets: http://www.data-mine.com/Products.htm
[42] PolyAnalyst: http://www.megaputer.com/pasystem.html
[43] PrudSys Discoverer: http://www.prudsys.com/discoverer
[44] Scenario: http://www.cognos.com/scenario/
[45] S-PLUS Professional: http://www.mathsoft.com/splus/
[46] SAS Enterprise Miner: http://www.sas.com/software/components/miner.html
[47] See 5 / C5.0: http://www.rulequest.com/see5-info.html
[48] SphinxVision: http://www.asoc.de/main2.html
[49] SRA KDD Explorer: http://www.knowledgediscovery.com/home/product/toolset.html
[50] Syllogic Data Mining: http://www.syllogic.nl/aboutsyllogic/productdataminingtool.html
[51] Viscovery SOMine: http://www.eudaptics.com/somine.htm
[52] VisualMine: http://www.visualmine.com/Datasheet/datasheet.htm
[53] XpertRule Miner: http://www.attar.com
[54] Snob: http://www.kdnuggets.com/software/clustering.html
(Ultimo acceso: 18/05/2000)

Otros
[55] www.crt.umontreal.ca. Casos de prueba (Cordeau, Gendreau and Laporte).

ANEXO I - ESTADO DEL ARTE DEL CLUSTERING

ANEXO II - SOFTWARE DE CLUSTERING.

ANEXO III – CASOS COMPLEMENTARIOS

ANEXO IV – MANUAL DE USUARIO

ANEXO V – DIAGRAMA DE GANTT