Capítulo 5

Metodología de Solución

Para resolver las tres tareas del problema de transportación RoSLoP: definición y asignación de rutas, asignación de horarios y asignación de cargas, se propone una metodología de solución de dos fases. En la primera sección de este capítulo se describen en forma general estos pasos. El resto del capítulo profundiza en la metodología, detallando las técnicas algorítmicas usadas en la solución RoSLoP.

5.1 Fases de la Metodología de Solución

La metodología propuesta para dar solución al problema de transporte RoSLoP se muestra en la Figura 5.1. En este diagrama se ven los tres subproblemas de RoSLoP: definición y asignación de rutas; asignación de cargas; y asignación de horarios.

La metodología de solución consta de dos fases. En la primer fase, la definición y asignación de rutas se resuelve con un algoritmo de Sistema de Colonia de Hormigas (ACS, de sus siglas en inglés Ant Colony System) basado en un enfoque Optimizador de Colonias de Hormigas (ACO, Ant Colony Optimization). Dentro de esta fase también se soluciona el

sub-problema de asignación de carga a través del algoritmo DiPro, una heurística determinista que sigue una estrategia round-robin para balancear los bienes distribuidos en el vehículo.

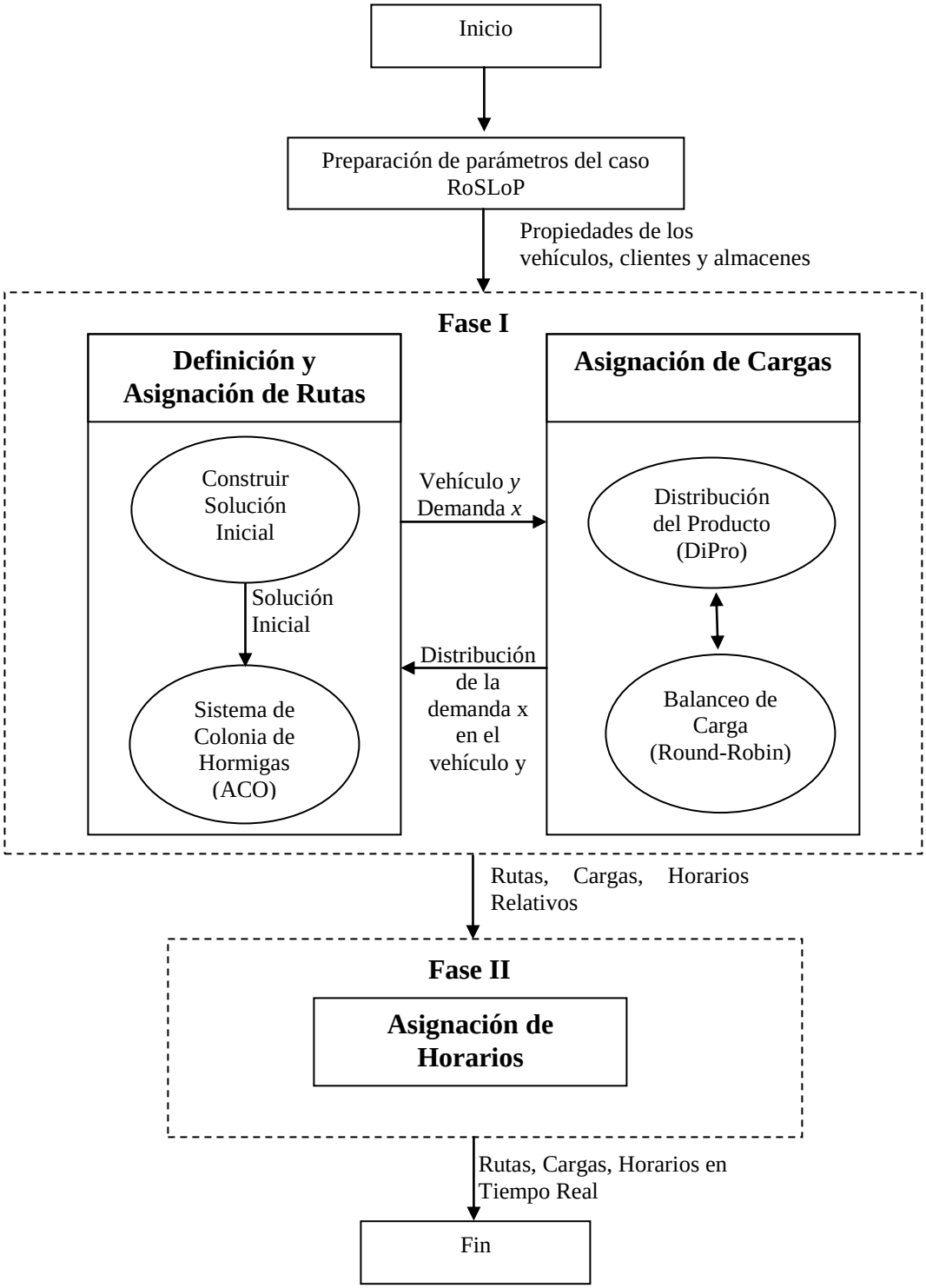


Figura 5.1. Metodología de Solución.

A fin de construir la solución es necesario crear rutas que visiten todos clientes; esta tarea se realiza a través del algoritmo ACS. Cada vez que una unidad de transporte visita un cliente, ACS invoca al algoritmo DiPro, que se encarga de determinar la parte de la demanda que transportará el vehículo. El resultado alcanzado en este primer paso contendrá las rutas y flotilla necesaria para abastecer los clientes.

Una vez que se han resuelto la asignación de rutas, y la asignación de cargas, se procede con el segundo paso de la metodología: la asignación de horarios. Debido a que el primer paso de la metodología ofrece un solución inicial al sub-problema de asignación de horarios (un conjunto de horarios relativos para el caso RoSLoP), en esta segunda fase solamente es necesario transformar esos horarios estándar a horarios en tiempo real requerido por el caso RoSLoP; lo cual se hace a través de una simple conversión.

En las siguientes subsecciones se analizarán con más detalle las herramientas heurísticas que sirvieron de base para la construcción de la metodología de solución.

5.2 Sistema de Colonia de Hormigas (ACS)

Ciertas hormigas, cuando regresan al nido con comida, dejan un rastro de feromona. Este rastro atrae y guía otras hormigas a la fuente de comida. Esto es continuamente renovado mientras la comida no se agote. Cuando sucede, los rastros dejados comienzan a cesar. La feromona se evapora rápidamente haciendo que las hormigas no se confundan por viejos rastros cuando la comida se encuentra en alguna otra parte.

El comportamiento cooperativo de las hormigas, mostrado en el párrafo anterior, es simulado en un Sistema de Colonia de Hormigas Artificial (ACS, Ant Colony System) y ha sido usado para resolver problemas NP-duros como el TSP en [Gambardella, 1996] o las variantes VRP en [Gambardella, 1999].

El algoritmo ACS diseñado para resolver la tarea de definición y asignación de rutas del problema RoSLoP está basado en los enfoques mostrados en [Gambardella, 1999] y

[Donati, 2003], y su meta es encontrar el mínimo número de vehículos necesarios para satisfacer todas las demandas de los clientes en los casos BPVRP de RoSLoP.

5.2.1 Análisis del Problema BPVRP resuelto por ACS

A fin de construir una solución, el algoritmo ACS propuesto usa dos medidas conocidas como cercanía y rastro de feromona. Debido a que el problema BPVRP incluye la variante HVRP, estas medidas miden no solo qué tan bueno es viajar entre un par de clientes (como se hace en [Gambardella, 1999]), sino también que tan adecuado es que un vehículo específico realice ese viaje.

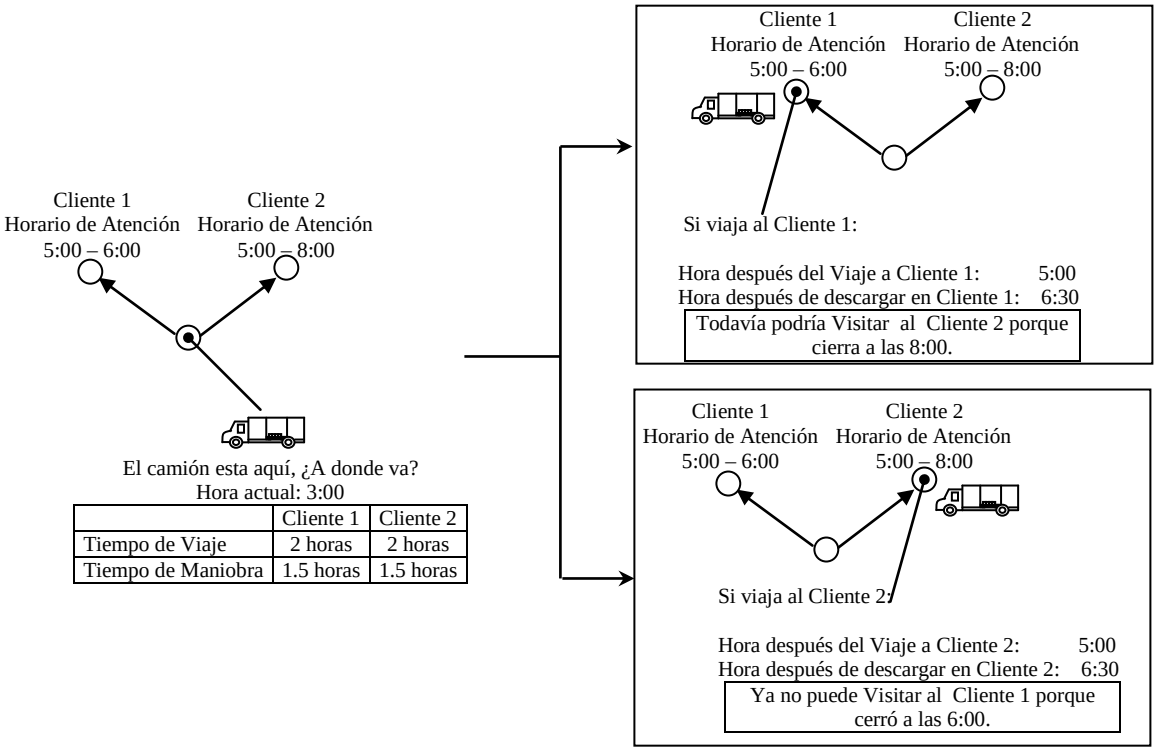
En el enfoque presentado en este trabajo de investigación, la cercanía η puede ser un valor asociado a clientes o vehículos. Estos valores son usados para indicar que tan bueno es viajar de un cliente a otro usando un vehículo previamente definido.

La cercanía entre dos clientes η_{ij} se calcula con la ecuación (5.1). En esta expresión, la calidad de un viaje de un cliente a otro depende del tiempo disponible para visitarlo.

$$\eta_{ij} = \frac{1}{\left(DeliveryTime_j - Now\right)\left(EndTimeWindow_j - Now\right)} \tag{5.1}$$

donde $DeliveryTime_j$ es el tiempo de llegada del vehículo actual al cliente j; Now es el tiempo en que el vehículo sale del cliente i para viajar al cliente j; y $EndTimeWindow_j$ es el tiempo en que termina la ventana de tiempo del cliente j.

El mejor cliente a visitar, de acuerdo a la ecuación (5.1), es aquel que si no es atendido en el momento actual ya no podría atenderse más tarde, tal vez por que llegaría al límite de su hora de atención. En la Figura 5.2 se ilustra un ejemplo de esta situación y su posible solución.



	Cliente 1	Cliente 2
Tiempo de Viaje	2 horas	2 horas
Tiempo de Maniobra	1.5 horas	1.5 horas

Figura 5.2. Ejemplo de selección del siguiente cliente a visitar. En este ejemplo la mejor opción es el Cliente 1, ya que visitarlo no impediría que el Cliente 2 se visitara después.

La ecuación (5.2) se utiliza para calcular la cercanía de un vehículo η_c**.** En otras palabras, esta ecuación hace que sea más atractivo el camión que maximice el número de clientes visitados por dicho vehículo.

$$\eta_c = \frac{1}{nv_c \cdot \left(\overline{TM_c} + \overline{TR_c}\right) \cdot \frac{tr_c}{tt_c}} \tag{5.2}$$

En la Tabla 5.1 se muestran las definiciones de los elementos necesarios para calcular la cercanía de los vehículos.

Tabla 5.1. Definición de los parámetros de la función de cercanía para vehículos.

Símbolo	**Definición**
η_c	Cercanía del vehículo *c*.
N_k	Conjunto de clientes con una demanda d_i a ser satisfecha por el almacén *k*.
C_k	Flotilla de vehículos del depósito *k*.
c	Un vehículo específico $c \in C_k$.

i	Un cliente $i \in N_k$
nv_c	Viajes requeridos por el vehículo $c \in C_k$ para abastecer todas las demandas d_i, para cada $i \in N_k$.
$\overline{TM_c}$	Tiempo promedio de Carga/Descarga de un vehículo $c \in C_k$.
$\overline{TR_c}$	Tiempo promedio de viaje de un vehículo $c \in C_k$.
tr_c	Tiempo de servicio disponible del vehículo $c \in C_k$.
tt_c	Tiempo de servicio total del vehículo $c \in C_k$.

Para evaluar la calidad de un vehículo, con el fin de ser seleccionado para integrarse a la solución, la ecuación (5.2) combina tres de sus propiedades: tamaño, velocidad y tiempo de servicio. El factor nv_c hace más atractivo a los camiones grandes, debido a que requieren menos viajes para cubrir todas las demandas. El factor $\overline{TM_c} + \overline{TR_c}$ favorece las unidades más rápidas, porque podrán cubrir un mayor número de rutas. Por último, el factor tr_c / tt_c selecciona aquellos camiones que ya hayan consumido parte de su tiempo de servicio cubriendo rutas previamente asignadas, esto se hace así para asegurarse que los vehículos sean empleados de forma exhaustiva. El vehículo que posea el mejor balance entre estas propiedades será el mejor de esta expresión.

El rastro de feromona τ es un valor que mide que tan deseable es incluir un elemento particular dentro de una solución. Como la cercanía η, la feromona está asociada con un par de clientes y con cada vehículo. En contraste, su valor es modificado por las hormigas durante la construcción de las soluciones.

La actualización global del rastro de feromona para los clientes τ_{ij} se muestra en la ecuación (5.3). La ecuación (5.4) muestra la actualización local de esta medida.

$$\tau_{ij} = (1-\rho)\cdot\tau_{ij} + \rho / J_{\psi}^{gb} \tag{5.3}$$

donde $\rho\left(0 \le \rho \le 1\right)$ es un parámetro de evaporación y J_{ψ}^{gb} es la longitud de la mejor solución global ψ^{gb} alcanzada por el algoritmo ACS al momento de la actualización.

$$\tau_{ij} = (1-\rho)\cdot\tau_{ij} + \rho\cdot\tau_0 \qquad (5.4)$$

donde:

τ_0 es el valor inicial del rastro y su valor es igual a $1/(n\cdot J_\psi^{gb})$. El valor *n* es el número de clientes satisfechos o vehículos empleados, según sea la ecuación donde se emplee.

La actualización local de la feromona de los vehículos τ_c se realiza durante la construcción de una solución con la ecuación (5.5).

La actualización global de la feromona de los vehículos τ_c se muestra en la ecuación (5.6). Los parámetros de las expresiones de actualización de feromonas son los mismos de las ecuaciones (5.3) y (5.4).

$$\tau_c = (1-\rho)\cdot\tau_c + \rho\cdot\tau_0 \qquad (5.5)$$

$$\tau_c = (1-\rho)\cdot\tau_c + \rho / J_\psi^{gb} \qquad (5.6)$$

La estrategia que construye una solución combina las estrategias de explotación y exploración tomadas de [Gambardella, 1999]. La explotación combina tanto el rastro de feromona τ como la cercanía η, para indicar la deseabilidad que un cliente o vehículo tienen para ser incluidos en una solución. En la ecuación (5.7), el elemento escogido *s* es el que maximice el valor de la función de explotación.

$$s = \max\{\tau\cdot[\eta]^\beta\}, \qquad s \in N^k \qquad (5.7)$$

En la ecuación (5.7), β pondera la relativa importancia de la cercanía η en la construcción de una solución y N^k es un conjunto de elementos factibles que puede ser incorporado a la solución que se está construyendo. Dentro de este conjunto se pueden identificar dos subconjuntos: el conjunto de vehículos factibles N_v^k y el conjunto de clientes factibles N_i^k donde $N_i^k \cup N_v^k = N^k$ y $N_i^k \cap N_v^k = \varnothing$. El conjunto N_v^k contiene todos los

vehículos con tiempo disponible para abastecer a los clientes mientras que N_i^k contiene los clientes que pueden ser visitados por un vehículo desde el cliente *i*.

En la exploración, la solución es construida en forma probabilística. Los elementos de N^k pueden ser escogidos con una probabilidad p que es proporcional a la ecuación (5.7). La ecuación (5.8) muestra como calcular la probabilidad para cada elemento *s* (clientes o vehículos) que pueden ser incorporados en la solución que se está construyendo.

$$p_s = \begin{cases} \dfrac{\tau_s \cdot [\eta_s]^\beta}{\sum_{e \in N^k} \tau_e \cdot [\eta_e]^\beta} & \text{si } s \in N^k \\ 0 & \text{en otro caso} \end{cases} \tag{5.8}$$

Para escoger una de las dos estrategias de construcción de soluciones se utiliza el parámetro q_0, que representa la probabilidad de uso de la técnica de explotación en la construcción de una solución. La probabilidad $1 - q_0$ se convierte en la oportunidad que tiene la exploración de ser elegida.

Regularmente los parámetros de control del algoritmo (τ_0, β, q_0, ρ) se determinan mediante experimentación.

5.2.2 Algoritmo

El algoritmo ACS se muestra en la Figura 5.3. Para construir una solución para un caso BPVRP, sigue el siguiente procedimiento general: en la línea 1 crea una solución inicial factible, usando el algoritmo clásico del vecino más cercano, que se tomará como el mejor global ψ^{gb}. En la línea 3 inicio ACS, que concluirá cuando el tiempo programado halla expirado (línea 22). De las líneas 4 a la 18 se encuentra el proceso de minimización de vehículos que creará la colonia un número específico de veces (línea 18). En éste proceso se busca mejorar la solución global y se actualizan las matrices de feromona local y global (líneas 14 a 17). Cuando una solución local ha sido mejorada (línea 9), se termina la colonia actual (línea 12), se compara la solución local con la global (línea 19) y se hacen las actualizaciones necesarias (líneas 19 y 20) para que seguir con el algoritmo.

ACS_Algorithm (β , ρ , q_0)

1. $\psi^{gb} \leftarrow$ Initial_Solution /* *Inicializar la solución global con una solución factible o infactible* */
2. $t \leftarrow$ #active_vehicles(ψ^{gb}) – 1 /* *Calcula el número de vehículos de la solución global* */
3. repeat /* *Comienza el algoritmo ACS* */
4. repeat /* *Comienza el proceso de minimización de vehículos* */
5. $\psi^{lb} \leftarrow$ Initial _Solution /* *Crea solución local inicial (factible o infactible)* */
6. for each ant $k \in K$ /* *Comienza el proceso de optimización basado en hormigas* */
7. $\psi^k \leftarrow$ new_ant_solution(k, t, IN) /* *Construye una nueva solución con v vehículos* */
8. $\forall j \notin \psi^k$: $IN_j \leftarrow IN_j + 1$ /* *Actualiza vector IN* */
9. if #visited_customers(ψ^k) > #visited_customers(ψ^{lb}) then
10. $\psi^{lb} \leftarrow \psi^k$
11. $\forall j$: $IN_j \leftarrow 0$
12. end_repeat *proceso de minimización de vehículos*
13. End for each
14. $\forall c \in \psi^{lb}$: $\tau_C = (1-\rho)\cdot\tau_C + \rho / J_\psi^{lb}$ /* *Actualiza feromona de vehículos con solución local* */
15. $\forall i, j \in \psi^{lb}$: $\tau_{ij} = (1-\rho)\cdot\tau_{ij} + \rho / J_\psi^{lb}$ /* *Acutaliza fermona de clientes con solución local* */
16. $\forall c \in \psi^{lb} : \tau_C = (1-\rho)\cdot\tau_C + \rho / J_\psi^{gb}$ /* *Actualiza feromona de vehículos con solución global* */
17. $\forall i, j \in \psi^{gb} : \tau_{ij} = (1-\rho)\cdot\tau_{ij} + \rho / J_\psi^{gb}$ /* *Actualiza feromona de clientes con solución global* */
18. Until stop criterion is reached
19. if ψ^{lb} is feasible and ψ^{lb} improves ψ^{gb} then /* *Si se mejora la solución global* */
20. $\psi^{gb} \leftarrow \psi^{lb}$
21. $T \leftarrow$ #active_vehicles(ψ^{gb}) – 1
22. until stop criterion is reached

Figura 5.3. Algoritmo ACS utilizado para resolver casos RoSLop.

El rastro de feromona se actualiza dos veces durante el desarrollo del algoritmo. Primero se actualiza durante la construcción de la solución por una hormiga y después de que todas las hormigas han construido sus soluciones

En el algoritmo, las funciones *#active_vehicles* y *#visited_customers* determinan el número de vehículos usados en una solución y el número de demandas satisfechas, respectivamente. La función *#depot* retorna a qué almacén pertenece el vehículo pasado como parámetro. El vector entero IN_j contiene el número de veces que un cliente j no se ha tomado en cuenta para formar una solución. IN es empleado por el proceso de construcción para favorecer a los clientes que no se incluyen frecuentemente en una solución Ψ.

5.2.3 Proceso de Construcción de Soluciones

El proceso constructivo *new_ant_solution* crea la nueva solución con la hormiga *k*(ver Figura 5.4); consiste en generar rutas asignadas a un vehículo *t* que satisfaga las demandas de los clientes que se encuentran en dicha ruta.

El primer paso para crear una ruta es determinar el conjunto de vehículos disponibles N_v^k (línea 2), que contendrá los vehículos que todavía pueden ser usados para abastecer clientes. Después, la cercanía de cada vehículo que pertenece a N_v^k es calculada (líneas 3 a 6) y entonces uno de ellos es seleccionado. Durante este paso se emplean los mecanismos de exploración y explotación.

Una vez que se ha escogido un vehículo (línea 7), el siguiente paso es visitar los clientes y satisfacer sus demandas (líneas 10 a 22). El vehículo sale del almacén para servir a los clientes (línea 9). La decisión de cual cliente debe visitarse primero se toma usando las estrategias de exploración y explotación (líneas 11 a 17). Mientras el conjunto N_i^k tenga clientes para visitar (línea 22) el vehículo seguirá abasteciéndolos, de otra forma regresará al depósito. Durante el proceso de construcción, la feromona se actualiza localmente.

El procedimiento para la creación de rutas se repite hasta que ya no se puedan incorporar más clientes a la solución o bien ya no puedan utilizarse más vehículos (línea 24).

En la función *new_ant_solution* se contempla el manejo de todas las variantes que son resueltas por nuestro algoritmo. En la línea 7 se maneja la variante VRPM; el conjunto N_v^k incluye aquellos vehículos que todavía tienen tiempo disponible para servir a los clientes, aún si los vehículos han sido asignados a otras rutas. La variante VRPTW está contemplada en el conjunto de clientes factibles N_i^k en la línea 11; este conjunto contendrá los clientes que pueden ser visitados por un vehículo específico de acuerdo a su ventana de tiempo. sdVRP es manejado en las líneas 7 y 11; el conjunto de clientes factibles N_i^k se construye

en la línea 11 al agregar aquellos clientes que pueden ser visitados por el vehículo *veh* escogido en la línea 7. La línea 19 contempla a VRPSD debido a que la actualización de la demanda de un cliente *j* significa que es posible que un vehículo que visite a *j* en una determinada ruta podría no satisfacer toda su demanda, dejando un resto a ser satisfecho por otra unidad. Las línea 22 maneja HVRP y CVRP al limitar la capacidad de los vehículos cuando se construye la ruta; la variante HVRP también necesita de la estructura *veh* que contiene las propiedades del vehículo entre las cuales se identifica el valor de su capacidad.

new_ant_solution(*k, t, IN*)

1. Loop
2. Determine the set of feasible vehicles N_v^k
3. For each vehicle $w \in N_v^k$
4. $\eta_w \leftarrow nv_w \cdot \left(\overline{TM_w} + \overline{TR_w}\right) \cdot \left(tr_w / tt_w\right)$
5. $\eta_w \leftarrow 1 / \eta_w$ */* Calcular la cercanía de los vehículos */*
6. End for each
7. Choose next vehicle *veh* $\in N_v^k$ using exploitation and exploration mechanisms
8. $\Psi \leftarrow \Psi \cup veh$ */* Incorporar vehículo seleccionado a la solución */*
9. $i \leftarrow$ #depot(*veh*) */* Establecer al almacén como la primer localidad visitada por veh */*
10. Loop
11. Determine the set of customers N_i^k the vehicle *veh* can visit from *I*
12. for each customer $j \in N_i^k$
13. $\eta_{ij} \leftarrow \left(DeliveryTime_j - Now\right)\left(EndTimeWindow_j - Now\right)$
14. $\eta_{ij} \leftarrow 1 / \eta_{ij}$ */* Calcular el valor de cercanía para los clientes */*
15. $\eta_{ij} \leftarrow$ max(1.0, η_{ij} - IN_j) */* Favorecer clientes poco visitados */*
16. end for each
17. Choose next customer $j \in N_i^k$ using exploitation and exploration mechanisms
18. $\Psi \leftarrow \Psi \cup j$ */* Incorporar clientes elegidos a la solución */*
19. Update demand of customer *j*.
20. $\tau_{ij} = (1-\rho) \cdot \tau_{ij} + \rho \cdot \tau_0$ */* Actualización local de la feromona del clientes */*
21. $i \leftarrow j$ */* Cambiar ubicación del vehículo actual al siguiente nodo */*
22. Until No more customers can be satisfied with *veh*
23. $\tau_{veh} = (1-\rho) \cdot \tau_{veh} + \rho \cdot \tau_0$ */* Actualización local de la feromona del cliente */*
24. Until Satisfy all customer demands or #active_vehicles(Ψ) > *t*
25. Return Ψ

Figura 5.4. Función *new_ant_solution*.

5.3 Algoritmo DiPro

Basado en la estrategia Round Robin, la cual es una estrategia muy conocida usada en sistemas operativos, el algoritmo DiPro distribuye el pedido de un cliente en los contenedores de los vehículos. Debido a que la demanda de los clientes está hecha en cajas de productos, el algoritmo usa P_c para representar el conjunto de productos que son demandados por un cliente c. Cada $p \in P_c$ tiene una cantidad q_{pc} que es la cantidad de producto p que ha sido pedida por el cliente c.

Las propiedades asociadas a cada producto son: cajas de producto por cama pbb_p, que contiene el número de cajas de producto p necesarias para formar una cama homogénea; camas de producto por tarima pbp_p, que es el número de camas homogéneas del producto p necesarias para formar una tarima homogénea; peso soportado sw_p del producto p, el peso que puede estar sobre una caja de este producto sin sufrir daño alguno; la altura h_p de una caja específica de producto p; peso w_p del producto p; categoría cat_p y área de resto ra_p. Las dos últimas propiedades son empleadas para mostrar la relación de p con otros productos; los productos compatibles a p tendrán el mismo valor en estas propiedades.

En la Figura 5.5 se muestra el Algoritmo DiPro. A fin de distribuir la demanda de un cliente P_c en un vehículo, los productos en P_c son ordenados por su peso soportado (línea 1). Después de eso, los bienes son organizados en tarimas homogéneas hom_plat_p y camas homogéneas hom_bed_p (líneas 2 a 7). Así, las cajas de producto restantes de cada producto rpb_p son usadas para formar camas heterogéneas het_bed_p con productos que tienen el mismo valor en las propiedades de categoría y área de restos (líneas 8 a 15). La variable het_bed_p contiene el mismo número de cajas que una cama homogénea debido a que de acuerdo a la compatibilidad de productos los bienes compatibles solo difieren en su peso o soporte pero no en dimensiones.

El resto de las camas homogéneas y heterogéneas que no pudieron formar tarimas homogéneas en el proceso previo, son combinadas para formar tarimas mezcladas het_plat_r por cada área de restos. La estrategia para formar esas nuevas tarimas tiene dos pasos. El primer paso consiste en calcular el número de tarimas necesarias por área de restos

needed_platforms$_r$ para acomodar las camas (líneas 17 a 20). En el segundo se construyen las tarimas mezcladas *het_plat*$_r$, usando la estrategia Round Robin (línea 21), al combinar las camas homogéneas y heterogéneas, de la misma área de restos.

DiProAlgorithm(P_c)
1. Order P_c by supported weight s_p
2. for each $p \in P_c$ /* *Calcular y construir camas y tarimas homogéneas* */
3. $hom\_bed_p \leftarrow q_{pc} / pbb_p$
4. $hom\_plat_p \leftarrow hom\_bed_p / pbp_p$
5. $hom\_bed_p \leftarrow hom\_bed_p$ Mod pbp_p /* *Determinar camas restantes de p* */
6. $rpb_p \leftarrow q_{pc}$ Mod pbb_p /* *Determinar cajas restantes de p* */
7. end for
8. for each $p \in P_c$ /* *Formar camas heterogéneas usando* rpb_p */
9. if $rpb_p \neq 0$ then
10. loop
11. search a $q \in P_c$ such that $cat_p = cat_q$ and $ra_p = ra_q$
12. $het\_bed_p \leftarrow$ sumar rpb_q a rpb_p
13. update rpb_p y rpb_q
14. while het_bed_p be incomplete
15. end for each
16. for each area de restos r /* *Construir tarimas mezcladas* */
17. for each $p \in P_c$ such that $ra_p = r$
18. $total\_beds\_left \leftarrow total\_beds\_left + hom\_bed_p + het\_bed_p$
19. end for each
20. $needed\_platforms_r \leftarrow total\_beds\_left / pallet\_height$
21. $het\_plat_r \leftarrow$ RoundRobin($needed\_platforms_r$, r)
22. end for each
23. $Platforms \leftarrow$ #DeterminePlatforms(hom_plat_p) $\cup$ #DeterminePlatforms(het_plat_r)
24. $total\_platforms \leftarrow |Platforms|$ /* *Calcula el número total de tarimas a distribuir* */
25. Order set *Platforms* by *weight* /* *Ordenar las tarimas por peso* */
26. for each $plat \in Platforms$ /* *asignar las tarimas al vehículo de forma balanceada* */
27. assign *plat* to the vehicle
28. balance the vehicle load /* *balancear los lados del vehículo y ubicar pesados adelante* */
29. end for each
30. assign remaining beds to available space /* *aprovechar espacios disponibles* */

Figura 5.5. Algoritmo DiPro.

Las función *#DeterminePlatforms* devuelve las tarimas asociadas a un tipo específico. Una vez que todas las tarimas *Platforms* han sido construidas e identificadas (línea 23), prosigue su asignación a contenedores (líneas 24 a 29). Para hacer esto, todas las tarimas son ordenadas por peso (línea 25) y después asignadas, tomando en cuenta que los lados,

tanto izquierdo como derecho, del camión estén balanceados en peso y además que las plataformas más pesadas se encuentren al frente y las más ligeras atrás (líneas 26 a 29).

Al terminar de establecer la distribución de las tarimas en el vehículo, se busca asignar los productos que no pudieron ser acomodados (línea 30), en el espacio disponible que no fue ocupado por otros bienes; debido a ciertas restricciones que no se cumplieron, como peso soportado o altura del contenedor.

Como ya se mencionó, el algoritmo DiPro usa la técnica Round Robin para construir las tarimas mezcladas. Un round-robin es un arreglo que se forma escogiendo elementos de un grupo, que se encuentran en un orden racional, normalmente del inicio al fin de la lista y luego volviendo a comenzar desde el principio, siguiendo así hasta haber seleccionado a todos. En DiPro, el arreglo se representa por las camas restantes de producto que no se pudo ubicar en tarimas homogéneas; el orden está dado por el peso soportado. El algoritmo Round Robin usado se muestra en la Figura 5.6. Este algoritmo consiste en crear tarimas vacías *needed_platforms*$_r$ suficientes para ubicar los camas restantes de un área de restos *r* (línea 2). Posteriormente, selecciona una cama a la vez, por orden de peso soportado (línea 4), y se agrega a una de las tarimas creadas (línea 5). Éste proceso se repite sucesivamente, cambiando de tarima en cada movimiento (línea 7) de manera que se repartan las camas equitativamente, hasta terminar de construir las tarimas mezcladas.

```
    RoundRobin(needed_platforms_r, r)
1.  Platform_Counter ← 1
2.  create needed_platforms_r heterogeneous platforms
3.  loop /* Construir tarimas mezcladas */
4.      search available bed b that have the greatest supported weight and that belong to r
5.      allocate b in the platform Platform_Counter
6.      mark bed b as no available
7.      Platform_Counter ← Next Platform_Counter
8.  until no more remaining beds can be allocated in the platforms
```

Figura 5.6. Algoritmo Round-Robin.

5.4 Conversión de Horarios

De acuerdo con la definición de VRPTW mostrada en [Dorronsoro, 2005], una ventana de tiempo es el horizonte de planeación en el cual un cliente puede ser servido. Una solución

para el VRPTW proporcionado por la primera fase de nuestra metodología incluye un horario relativo al tiempo real exigido por el caso RoSLoP. Debido a esto, el único trabajo requerido, una vez que ha finalizado el paso uno, es la conversión del horario relativo al real establecido por los clientes que se van a abastecer. La conversión consiste en tomar cada uno de los horarios producidos en la solución de VRPTW y transformarlo en el horario real. El número de horarios máximo que se puede transformar en un caso de RoSLoP es equivalente a decir que cada cliente es visitado por todos los vehículos disponibles; es decir, sea m el número de vehículos y n el número de clientes, entonces el número de horarios en el peor de los casos es igual a $m \cdot n$.