Universidad de La Habana
Facultad de Matemática y Computación

Generación automática de gramáticas para la obtención de infinitos criterios de vecindad en el problema de enrutamiento de vehículos

Autor: **Daniela González Beltrán**

Tutor: **MSc. Fernando Raúl Rodríguez Flores**

Trabajo de Diploma
presentado en opción al título de
Licenciado en Ciencia de la Computación

Mayo de 2019

A mi abuela ...

Agradecimientos

Me gustaría agradecer en primer lugar y de forma especial a mis padres que con tanta paciencia me brindaron todo su apoyo y dedicación. A toda mi familia por su apoyo desde siempre. En especial a mi madre por ayudarme con mis primeras clases prácticas.

A Marcel por aguantar tantas locuras, llantos y gritos y por todas las horas que me acompañó en el desarrollo de este trabajo y de muchos otros. A Floopy por secarme las lágrimas y alegrarme los días.

A Claudia Elvira, Laura, Sherlhy y Gabriela por su amistad, apoyo y por tantas horas de estudio. A Michael por ayudarme con mis primeros programas.

A mi tutor Fernando por tener tanta paciencia y proponerme este trabajo. A todos mis profesores en especial a Karel, José Javier, Nayla, Camila y Aymée.

Opinión del tutor

El problema de enrutamiento de vehículos (VRP) ha sido estudiado durante más de 50 años, en la literatura se pueden encontrar más de 30 variantes, y constantemente están apareciendo nuevos problemas y nuevas vías de solución. Con el trabajo realizado en esta tesis es posible obtener, con muy poco esfuerzo humano, infinitos criterios de vecindad para resolver "cualquier" VRP.

Para lograrlo, Daniela tuvo que incursionar en temas de la matemática y la computación que no forman parte de su plan de estudio como el problema de enrutamiento de vehículos, y paradigmas y estilos de programación muy diferentes a los conocidos por ella. Durante este tiempo de trabajo, fue capaz, no solo de resolver los problemas planteados, sino de también de identificar (y resolver de manera independiente) nuevos problemas, que aparecieron durante el proceso.

Estoy muy contento con los resultados obtenidos. Considero que estamos en presencia de un trabajo excelente y que el mismo es una evidencia de que Daniela reúne todos los requisitos para desempeñarse como una excelente científica de la computación.

MSc. Fernando Raul Rodriguez Flores
Facultad de Matemática y Computación
Universidad de la Habana
Mayo, 2019

Resumen

Los problemas de enrutamiento de vehículos (VRP), engloban a una familia de problemas de optimización combinatoria. La naturaleza NP-Duro de los VRP ha conducido al empleo de metaheurísticas para solucionarlos, y en particular las de búsqueda local. Estos algoritmos necesitan el empleo de criterios de vecindad. Es posible considerar infinitos criterios de vecindad mediante gramáticas libre del contexto. Cada variante del VRP requiere una gramática distinta, y hacer este trabajo manualmente requiere tiempo. En esta investigación se propone una representación de los problemas VRP y un mecanismo que, a partir de dicha representación, obtiene de forma automática una gramática que genera infinitos criterios de vecindad. Esta propuesta puede contribuir a reducir el tiempo empleado por los investigadores evitando la construcción de las gramáticas de forma manual.

Abstract

The vehicle routing problem (VRP) encompasses a family of several combinatorial optimization problems.Given the NP-hard nature of vehicle routing problems and the complexity of solving to optimality large scale instances of them, a significant research effort has been dedicated to metaheuristics, in particular those related to local search. These algorithms rely on the use of neighborhood structures. It is possible to consider infinite neighborhood structures through context free grammars. Each variant of VRP requires a specific grammar and doing this entire process manually requires large amounts of time. This investigation presents a representation of VRP and a mechanism based on said model that automatically obtains a grammar capable of generating infinite neighborhood criteria. This proposal can potentially reduce the time employed by investigators, thus avoiding having to construct the grammars manually

Índice general

Introducción

Bajo el nombre Problema de Enrutamiento de Vehículos (VRP) se engloba una familia de problemas de optimización combinatoria [1] de gran importancia en el área de la logística [2]. Su solución requiere satisfacer la demanda de un conjunto de clientes distribuidos geográficamente minimizando un objetivo, que es específico para cada problema. Para ello cuenta con una flota de vehículos que parte de uno o varios depósitos centrales. El problema consiste, en el caso más sencillo, en asignar a cada vehículo un conjunto de clientes de modo que todos los clientes sean visitados una única vez.

Hasta la actualidad se han publicado más de 30 variantes del VRP [3], cada uno con especificaciones propias. Entre las más conocidas figuran las variantes con restricciones de capacidad (CVRP), con múltiples depósitos (MDVRP) y con ventanas de tiempo (VRPTW) [1]. Cada año aparecen nuevas variantes, que pueden estar motivadas por necesidades de la industria y el crecimiento de la sociedad [3, 4, 5], o por el desarrollo de investigaciones académicas [6].

En los últimos años, en la Facultad de Matemática y Computación de la Universidad de La Habana (MATCOM) se han solucionado numerosas variantes del VRP [5, 7, 8, 9]. A partir de la experiencia adquirida en la resolución de estos problemas, puede decirse que el tiempo requerido para ello oscila entre 6 meses y varios años. Si el problema se aborda en una tesis de licenciatura el tiempo estimado es de 6 meses [8, 9]; 1 o 2 años si la solución se presenta como resultado de una tesis de maestría [5, 7] y varios años si es el resultado de una tesis de doctorado [10].

En MATCOM se desarrolla actualmente una investigación que tiene como objetivo agilizar el proceso de solución de los problemas VRP para reducir el tiempo y esfuerzo humano dedicados. Estas reducciones se basan en que para encontrar una solución hay que realizar varios pasos mecánicos

que, por lo general, son los que más tiempo consumen durante la investigación. Algunos de estos pasos son: seleccionar la metaheurística con la que se resolverá el problema, pues usualmente son NP-duros [1]; elegir criterios de vecindad, cruzamiento o mutación, en dependencia del algoritmo escogido; y evaluar el comportamiento de la combinación algoritmo-criterio empleada. Estos pasos pueden ser automatizados utilizando lenguajes formales.

La propuesta de solución actual se basa en definir gramáticas cuyas cadenas puedan ser interpretadas como las posibles opciones para los pasos que se desean automatizar. Es decir, una gramática será la encargada de diseñar la metaheurística a utilizar y otra generará los criterios de vecindad, cruzamiento o mutación. A partir de las cadenas proporcionadas por cada una de estas gramáticas, es posible generar el código que ellas representan y evaluar cuán bien funcionan para el problema que se desea resolver. Esta idea fue utilizada por primera vez, en nuetra facultad, en el año 2017 [11].

El trabajo "Primeras aproximaciones a la Búsqueda de Vecindad Infintamente Variable"(IVNS) [11] propone una modificación a la metaheurística Búsqueda de Vecindad Variable (VNS), en la que se consideran infinitos criterios de vecindad. En este trabajo se diseñó una gramática cuyas cadenas se pueden interpretar como criterios de vecindad para el problema CVRP. Si se quisiera aplicar IVNS a otra variante del VRP, sería necesario diseñar otra gramática para este nuevo problema.

El objetivo general de este trabajo es proponer una estrategia que permita obtener de forma automática una gramática para la generación de infinitos criterios de vecindad para un problema VRP cualquiera.

Para alcanzar este objetivo general se proponen los siguientes objetivos específicos:

- Estudiar las distintas variantes del problema VRP analizando sus aspectos comunes y deferencias fundamentales.
- Proponer una forma de describir los problemas VRP en función de las operaciones básicas disponibles para la construcción de criterios de vecindad.
- Proponer una estrategia para, en función de la descripción, obtener una gramática libre del contexto que permita obtener infinitos criterios de vecindad.

- Implementar la estrategia propuesta como una herramienta extensible para la investigación en el ámbito de los problemas VRP.

Para dar solución al problema planteado, en la presente investigación se presenta una propuesta para describir las operaciones básicas que pueden utilizarse en los criterios de vecindad para un problema de enrutamiento de vehículos. Esta descripción esta basada en la especificación de las operaciones que conforman un criterio de vecindad y en las relaciones que existen entre estas operaciones. A partir de esta descripción es posible obtener la gramática que generar infinitos criterios de vecindad.

Este documento está estructurado de la siguiente forma: en el capítulo 1 se presentan los problema de enrutamiento de vehículos, los criterios de vecindad, los lenguajes formales y una breve introducción al lenguaje de programación *Lisp*. En el capítulo 2 se presenta una propuesta de descripción para las operaciones que pueden emplearse en los criterios de vecindad para un VRP. En el capítulo 3 se muestra como a partir de la descripción de las operaciones que pueden emplearse en los criterios de vecindad de un VRP se obtiene la gramática que genera infinitos criterios de vecindad para el problema que se analiza. En el capítulo 4 se presentan algunos detalles de la implementación de la estrategia propuesta. Finalmente se presentan las conclusiones y la bibliografía consultada.

Capítulo 1

Preliminares

En este capítulo se presentan los elementos fundamentales de este trabajo. En la sección 1.1 se presenta el Problema de Enrutamiento de Vehículos (VRP), así como algunas de sus variantes y vías de solución. En la sección 1.2 se presentan los elementos de la teoría de lenguajes formales necesarios para representar y generar criterios de vecindad para un VRP. En la sección 1.3 se introducen brevemente el concepto criterio de vecindad. Por último, en la sección 1.4 se presentan los elementos del lenguaje de programación Lisp relevantes para el desarrollo de este trabajo.

1.1. Problema de Enrutamiento de Vehículos

Uno de los primeros trabajos referentes al Problema de Enrutamiento de Vehículos (VRP) data de 1959, cuando Dantzing y Ramser [12] abordan un problema concerniente a la distribución de gasolina. En este artículo los autores presentan la formulación matemática del VRP y una aproximación algorítmica del mismo.

El objetivo del VRP, en su versión más simple, es diseñar un conjunto de recorridos para satisfacer la demanda de un conjunto de clientes dispersos geográficamente. Para ello se cuenta con una flota de vehículos que parten desde un depósito central. La solución del problema consiste en asignar a cada vehículo un conjunto ordenado de clientes, de modo que el costo total de transportación sea mínimo. Con la incorporación de restricciones y variables a este problema base, surgen las diferentes variantes del VRP. A continuación, se presentan algunas de estas.

1.1.1. Variantes de VRP

En este epígrafe se presentan cuatro variantes que poseen características distintas que hacen que los criterios de vecindad que pueden emplearse en cada una de ellas sean diferentes. Estas variantes son: VRP con restricciones de capacidad (CVRP), VRP con flota heterogénea (VRPH), VRP con múltiples depósitos y VRP con demanda dividida.

VRP con restricciones de capacidad (CVRP)

El CVRP es la variante más estudiada del VRP hasta la actualidad [1]. Consiste en la distribución de cierta cantidad de mercancía desde un depósito central a un conjunto de clientes. Se asume la existencia de una flota infinita y homogénea de vehículos, es decir, se pueden utilizar tantos vehículos como se desee y todos tienen la misma capacidad. Durante el recorrido, no se debe exceder la capacidad de los vehículos [2].

VRP con Flota Heterogénea (HVRP)

El HVRP es una variante donde la flota de vehículos no es homogénea. En este caso existen vehículos de diferentes características. Este problema se divide en dos tipos, en dependencia de si la cantidad de vehículos es finita (VRPHE) [3] o infinita [13].

VRP con múltiples depósitos (MDVRP)

En el MDVRP existen múltiples depósitos, por lo que los vehículos pueden iniciar su recorrido en diferentes depósitos, y esta decisión es parte de la solución del problema. Existen dos variantes de este problema: una primera en la que los vehículos están obligados a regresar al depósito donde inician su recorrido [13] y una segunda variante en la que esta restricción no se considera [14].

VRP con demanda dividida (SDVRP)

El SDVRP es una relajación del VRP en la que se permite que los clientes sean visitados por más de un vehículo. Esta relajación es de gran importan-

cia cuando la demanda del cliente puede ser mayor que la capacidad de los vehículos [3].

El VRP constituye una generalización del problema del viajante, por lo que constituye un problema NP-Duro [1]. Por esta razón, la literatura reporta soluciones a través de métodos exactos para instancias pequeñas, mientras que para problemas de mayores dimensiones se han usado heurísticas y metaheurísticas. En la próxima sección se presentan algunos de estos métodos.

1.1.2. Estrategias de soluciones para el VRP

En la literatura se reportan varios enfoques para resolver el VRP. Se han utilizados algoritmos exactos para resolver instancias pequeñas del problema. Laparote y Nobert [15] y Baldacci, Mingozzi y Roberti [16] presentan algoritmos de corte y ramificación (Branch and Cut), basados en formulación de particiones (set partitioning formulation) y programación dinámica.

Para instancias de grandes dimensiones se recomienda emplear heurísticas y metaheurísticas, ya que estos problemas son NP-duros [1]. Gendreau, Potvin, Bräumlaysy, Hasle y Løkketangen reportan en [17] el uso de metaheurísticas como Algoritmos Genéticos, Colonia de Hormigas, Búsqueda Adaptativa Aleatoria Golosa (GRASP) y metaheurísticas de Búsqueda Local como Recocido Simulado, Búsqueda Tabu y Búsqueda de Vecindad Variable (VNS).

Las metaheurísticas de Búsqueda Local reportan buenos resultados en la literatura para resolver el VRP. Para utilizar este tipo de metaheurísticas es necesario una solución inicial (que se considera la solución actual) y una estructura de entorno (o un conjunto de estructuras de entorno). En cada iteración, se explora el entorno de la solución actual y se selecciona un vecino. En dependencia de la metaheurísticas usada, y de si ese vecino mejora o no la evaluación de la función objetivo, la solución actual se remplaza por el vecino [18]. Por ejemplo, en recocido simulado se aceptan soluciones con peor evaluación de la función objetivo para evitar los mínimos locales; mientras que búsqueda tabú se rechazan soluciones con determinadas características (que se consideran tabú) [18].

Uno de los factores que más influyen en la calidad de los resultados de los algoritmos de búsqueda local son las estructuras de entornos, o criterios de vecindad. Uno de los objetivos de este trabajo es obtener infinitos crite-

rios de vecindad para cualquier problema VRP. Pérez Mosquera propone en [11] la obtención de estos criterios de vecindad a partir de las cadenas de un lenguaje formal. Esta idea se emplea en la presente invetigación, por lo que a continuación se presentan los elementos básicos de la teoría de lenguajes formales.

1.2. Teoría de Lenguajes Formales

Los conceptos básicos de la teoría de leguajes formales son: alfabeto, cadena y lenguaje. Un alfabeto es un conjunto finito y no vacío de símbolos. Una cadena es una secuencia finita de símbolos de un determinado alfabeto. Un lenguaje es un conjunto de cadenas cuyos símbolos provienen de un mismo alfabeto [19].

Una gramática libre del contexto es una cuádrupla $G = (N, T, P, S)$ donde N es el conjunto de símbolos no-terminales, T es el conjunto de símbolos terminales y ambos cumplen que $N \cap T = \emptyset$. P es el conjunto de producciones de la gramática, estas producciones tiene la forma $A \to \alpha$, $A \in N$ y $\alpha \in (N \cup T)^*$. $S \in N$ es un símbolo no terminal, llamado símbolo inicial.

Como convenio en la literatura, los símbolos no terminales de la gramática se representan con letras mayúsculas y los símbolos terminales con letras minúsculas. Se asume que símbolo inicial es siempre S. Teniendo en cuenta estos convenios es posible representar las gramáticas a través del conjunto de producciones.

Una forma oracional es una secuencia de símbolos terminales y no terminales. Una oración es una secuencia de símbolos terminales. Por ejemplo, sean los conjuntos $N = \{S\}$ y $T = \{a, b\}$ el conjunto de símbolos no terminales y el conjunto de símbolos terminales respectivamente. La cadena aSb es una forma oracional ya que posee símbolos terminales y no terminales. La cadena ab es una oración ya que solo posee símbolos terminales.

Un lenguaje formal puede tener una cantidad finita o infinita de cadenas. En caso de que un lenguaje sea infinito, se puede representar de una manera finita utilizando una gramática.

Por ejemplo el lenguaje cuyas cadenas tienen la forma

$$a^n b^n \text{ donde } n \in \mathbb{N}$$

es infinito y como se puede ver en la figura 1.1 la gramática que genera este

lenguaje tiene solo dos producciones.

$$S \rightarrow aSb$$
$$S \rightarrow ab$$

Figura 1.1: Gramática que genera el lenguaje $a^n b^n$

El lenguaje que representa los criterios de vecindad de un VRP es infinito. Con el objetivo de representar de forma finita estos lenguajes en este trabajo se emplean gramáticas libres del contexto. Para ello los criterios de vecindad deben ser representados como cadenas. En el siguiente epígrafe se define que es un criterio de vecindad y cómo puede ser representado a través de una cadena de un lenguaje formal.

1.3. Criterios de Vecindad

Para poder definir un criterio de vecindad es necesario definir otros dos conceptos. El primero es qué se entiende por solución de un problema y el segundo qué es una operación básica.

La solución de un VRP esta compuesta por un conjunto de rutas y, en algunos casos, por conjuntos de elementos que pueden formar parte de las rutas. Cada ruta tiene asociado un conjunto de elementos. En la mayoría de los casos este conjunto está formado por clientes. Según las características del problema la composición de las rutas puede variar.

Por ejemplo: si en el problema la flota de vehículos es heterogénea cada ruta tiene asignado, además, el vehículo que trasporta la mercancía a los clientes asignados a dicha ruta, y como parte de la solución se tiene también el conjunto de vehículos que no están asignados a ninguna ruta. Si en el problema existieran múltiples depósito, a cada ruta se le asignaría, además, un depósito. Una solución para este problema contaría además con el conjunto de depósitos existentes.

El segundo concepto que se debe definir es operación básica. Estas son pequeñas modificaciones que se le pueden hacer a una solución. Por ejemplo, *seleccionar un cliente de una ruta* e *insertar un cliente seleccionado en una ruta* son operación básicas.

Una vez definido qué es una solución y qué es una operación básica, se puede definir un criterio de vecindad como una secuencia de operaciones básicas que se le aplican a una solución, de modo que el resultado de esas modificaciones siga siendo una solución válida para el problema.

Ejemplos de criterios de vecindad son *Mover un cliente de una ruta a otra*, *Reubicar un cliente dentro de su ruta* y *Intercambiar dos clientes*. Dado que un criterio de vecindad es una secuencia de operaciones básicas, los criterios antes mencionados pueden descomponerse de la siguiente forma:

Mover un cliente de una ruta a otra:

1. Seleccionar una ruta
2. Seleccionar un cliente de la ruta seleccionada
3. Seleccionar una ruta
4. Insertar el cliente seleccionado en la ruta seleccionada

Reubicar un cliente dentro de su ruta:

1. Seleccionar una ruta
2. Seleccionar un cliente de la ruta seleccionada
3. Insertar el cliente seleccionado en la ruta seleccionada

Intercambiar dos clientes:

1. Seleccionar una ruta
2. Seleccionar un cliente de la ruta seleccionada
3. Seleccionar una ruta
4. Seleccionar un cliente de la ruta seleccionada
5. Intercambiar dos clientes

Utilizando las operaciones básicas es posible construir un lenguaje formal, en el que cada cadena represente un criterio de vecindad. Para ello las operaciones básicas se deben representar como símbolos de un alfabeto. Una posible representanción para las operaciones utilizadas anteriormente es la siguiente:

- Seleccionar una ruta (r)
- Seleccionar un cliente de una ruta (a)
- Insertar un cliente en una ruta (b)
- Intercambiar dos clientes (c)

De esta forma es posible representar los criterios de vecindad *Mover un cliente de una ruta a otra Reubicar un cliente dentro de su ruta* e *Intercambiar dos clientes* con las cadenas *rarb*, *rab* y *rarac*, respectivamente.

La implementación actual del proceso de exploración de la vencindad requiere que las cadenas de los lenguajes formales cumplan una determinada restricción. Para expresar esta restricción se definen a continuación dos tipos de operaciones: operación de selección y operación de inserción.

Definición 1 *Una operación es de selección si selecciona al menos un elemento.*

Algunos de los elementos que se pueden seleccionar son rutas, clientes, depósitos, vehículos, entre otros. Las operaciones *Seleccionar una ruta* (r) y *Seleccionar un cliente de una ruta* (a) son operaciones de selección, ya que seleccionan rutas y clientes respectivamente.

Definición 2 *Una operación es de inserción si inserta al menos un elemento.*

La operación *Insertar un cliente en una ruta* (b) es una operación de inserción ya que inserta clientes. La operación *Intercambiar dos clientes* (c) está formada por la inserción de dos clientes intercambiando sus posiciones, por esta razón es una operación de inserción.

Una vez definidos los dos tipos de operaciones es posible plantear la restricción que deben cumplir las cadenas del lenguaje formal que representa los criterios de vecindad. Dichas cadenas deben cumplir que primero aparecen todas las operaciones de selección y luego todas las de inserción.

La cadena *rabrab* no pertenece al lenguaje, ya que existe una operación de inserción (b) que aparece antes que varias de las operaciones de selección (ra).

Dado que cada variante del VRP presenta características propias, no es posible emplear los mismos criterios de vecindad en cada una de las variantes. Por ejemplo, el criterio de vecindad *Intercambiar los vehículos de dos rutas*

es posible emplearlo en las variantes de VRP donde la flota de vehículos es heterogénea. Este criterio no tiene sentido en variantes del VRP dende la flota es homogénea porque al ejecutar este criterio se obtendría la misma solución.

El conjunto de operaciones básicas aplicables a un problema se obtiene a partir de la modelación del mismo. Dicha modelación se realizó en el lenguaje de programación *Common Lisp*. En la siguiente sección se presentan los elementos y las características de dicho lenguaje que fueron utilizadas.

1.4. Introducción a LISP

Las variantes de VRP presentan características que permiten diferenciarlas entre sí, pero también poseen características comunes. Por ejemplo, la mayoría de los VRP deben satisfacer la demanda de un conjunto de clientes. Existen variantes en las que todos los clientes deben ser visitados una única vez, otras en las que los clientes pueden ser visitados por más de un vehículo y otras en las que no todos los clientes deben ser visitados.

Estas características pueden representarse a través de una jerarquía de clases, de modo que se puede definir un VRP heredando de algunas de estas caraterísticas. *Common Lisp*, y en especial su sistema de objetos CLOS, presenta características como la herencia múltiple que facilitan el desarrollo de esta estrategia.

En el epígrafe 1.4.1 se presentan los elementos característicos de la sintaxis de *Lisp*, así como las características fundamentales de las funciones y macros. En el epígrafe 1.4.2 se presentan los elementos y características fundamentales del sistema de objetos de *Common Lisp* (CLOS).

1.4.1. Sintaxis, funciones y macros

Las expresiones en LISP se denominan s-expresiones. Estas están compuestas por listas y átomos. Un átomo está formado por caracteres concatenados que pueden ser números, letras o caracteres especiales. Las listas están delimitadas por paréntesis y pueden contener cualquier número de elementos separados por espacios en blanco. Los elementos de las listas son en sí s-expresiones (en otras palabras, átomos o listas anidadas). Las llamadas a funciones y macros son listas, con el nombre de la función o la macro como primer elemento, el resto de los elementos son los parámetros.

Las macros brindan al programador una forma de extender la sintaxis. Una macro es una función que toma s-expresiones como argumentos y devuelve código Lisp que será evaluado en donde tuvo lugar el llamado a la macro [20].

En la implementación de la estrategia propuesta se definen un conjunto de métodos que son muy similares. Para evitar la repetición de código se emplean las macros. En el capítulo 4 se brindan los detalles de la implementación de la macro y de los métodos antes mencionados.

Las carcterísticas que poseen las clases y las jerarquías de clases en el lenguaje *Common Lisp* que fueron utilizadas se exponen en el siguiente epígrafe.

1.4.2. Common Lisp Object System (CLOS)

El sistema de objetos de *Common Lisp* (CLOS) está formado por clases y funciones genéricas. Estas funciones permiten polimorfismo en múltiples argumentos y combinaciones de métodos [20]. A continuación se presentan estos elementos.

Clases

Los objetos se representan en CLOS a través de las clases. Las clases pueden contener campos, que están formados por un nombre y un conjunto de opciones que modifican su comportamiento. Una subclase es una clase que hereda los campos y el comportamiento de su superclase. CLOS soporta herencia múltiple, permitiendo a una clase tener más de una superclase.

Las diferentes características que pude presentar un VRP pueden representarse a través de una jerarquía de clases. La figura 1.2 muestra una jerarquía de clases. Para representar la carcterística de que en el problema existen rutas se define la clase *there-are-routes*. La clase *client-partition* representa que los clientes deben ser visitados una sola vez. heredando de esta clase se define el problema con restricciones de capacidad (CVRP). Para indicar que la flota de vehículos es heterogénea y finita se utiliza la clase *heterogeneous-finite-fleet*, y heredando de esta clase y de *client-partition* se define el VRP con flota heterogénea y finita (VRPHE).

La principal diferencia de CLOS con los sistemas de objetos tradiciona-

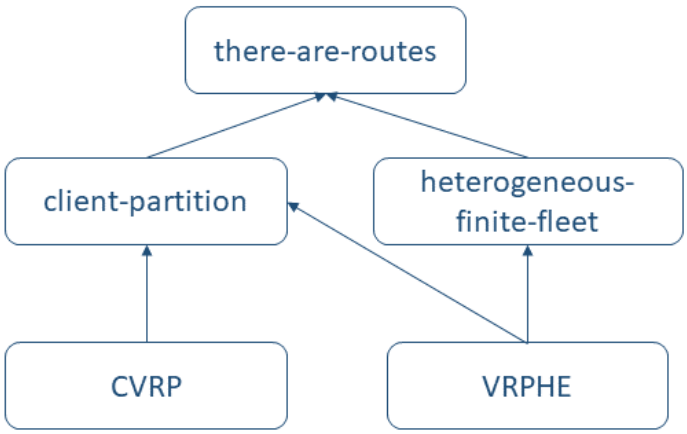

Figura 1.2: Jerarquía de clases

les es que en Lisp, los métodos no pertenecen a las clases sino a funciones genéricas. La siguiente sección aborda este tema.

Funciones genéricas

Una función genérica es un conjunto de métodos los cuales tienen el mismo nombre y número de parámetros. La diferencia entre los métodos asociados a una función genérica es el tipo que deben tener los parámetros. Al llamar a una función genérica se ejecuta el método cuyos parámetros sean más específicos con respecto al tipo de los argumentos.

Cuando se llama a una función genérica con unos argumentos específicos, el orden en que se llaman los métodos se modifica en dependencia de la combinación de métodos asociada a esa función genérica. El siguiente epígrafe está dedicado a los diferentes tipos de combinaciones de métodos que existen en *Common Lisp*.

Combinación de métodos

Los métodos pueden aumentarse mediante métodos auxiliares. En CLOS existen tres tipos de métodos auxiliares: "antes de" (*before method*),"después de" (*after method*) y "en lugar de" (*around method*). Los métodos auxiliares se definen al colocar una palabra clave (*before*, *after* and *around*) después del

nombre del método.

Lo que normalmente se conoce como método, en *Lisp* son métodos primarios. Como su nombre indica, los métodos “antes de” y “después de” se ejecutan antes y después que el método primario. Mientras que, si existe un método around, se llamará este en lugar del método primario.

Al orden en que se ejecutan estos métodos se le denomina combinación de métodos estándar. En la combinación de métodos estándar, al llamar a una función genérica se ejecuta:

1. El método *around* más específico, si existe.
2. De lo contrario se ejecuta, en orden:
   a) Todos los métodos “antes de", de más específico a menos específico.
   b) El método primario más específico
   c) Todos los métodos “después de", de menos específico a más específico.

En la combinación de métodos estándar, el único método primario al que se llama es el más específico (aunque puede llamar a otros a través del método *call-next-method*). En CLOS es posible combinar los resultados de todos los métodos primarios aplicables.

La combinación de métodos-operador puede entenderse como el resultado de la evaluación de una expresión de Lisp cuyo primer elemento es algún operador y cuyos argumentos son llamados a los métodos primarios aplicables, en orden de más específico a menos específico. Si definiéramos la función genérica de *price* para combinar valores con el operador $+$, y no existieran métodos *around* aplicables, se comportaría como si se definiera:

```
1 (defun price (&rest args)
2   (+ (apply (método primario más específico) args)
3      .
4      .
5      .
6      (apply (método primario menos específico) args)))
```

Si existen métodos *around* aplicables, estos tienen prioridad al igual que en la combinación de métodos estándar. En la combinación de método de

operador, un método *around* puede llamar al siguiente método aplicable a través de *call-next-method*. Sin embargo, los métodos primarios no pueden usar el método *call-next-call*.

A través de las característica del lenguaje *Common Lisp* explicadas en esta sección es posible representar los elementos que describen un problema de enrutamiento de vehículos. En el próximo capítulo se presenta una forma de describir un VRP a través de la cuál es posible obtener la gramática que genera infinitos criterios de vecindad para dicho problema.

Capítulo 2

Descripción de operaciones básicas para un VRP

Para cada variante del VRP es posible definir un lenguaje formal para representar los criterios de vecindad que pueden aplicarse en dicho problema. Por esta razón cada variante del VRP requiere una gramática distinta para generar el lenguaje de los infinitos criterios de vecindad. Dicha gramática se obtiene a partir de la descripción de las operaciones básicas de la variante de VRP que se analiza.

En este capítulo se presentan los aspectos que permiten caracterizar las operaciones básicas que pueden emplearse en los criterios de vecindad para un VRP. La descripción de las operaciones básicas que se propone en este trabajo, se divide en tres partes:

1. Especificación de operaciones básicas
2. Clasificación de operaciones básicas
3. Especificación de reglas

Primero debe especificarse cuáles son las operaciones básicas que se pueden utilizar en un criterio de vecindad para el problema en cuestión, ya que existen operaciones que solo tienen sentido en algunas variantes del VRP. Seguidamente, se determina para cada operación si requiere de la previa selección de una ruta y, en caso de necesitarlo, si más de una instancia de esa operación pueden utilizar la misma ruta seleccionada. Por

último, se definen la secuencia de operaciones que debe ejecutarse antes de cada operación de inserción. A continuación, se profundizará en cada uno de estos aspectos.

2.1. Especificación de operaciones básicas

Cada una de las variantes de VRP presenta características que la diferencia del resto. Por ejemplo, en cada variante del VRP se supone la existencia de un único depósito o de múltiples depósitos; la flota de vehículos puede ser finita o infinita, homogénea o heterogénea, etc. Por esta razón el primer paso de la descripción de las operaciones básicas es especificar cuáles son las operaciones que se pueden emplear en criterios de vecindad para el problema que se analiza.

En el CVRP las posibles operaciones son:

- Seleccionar una ruta (r)
- Seleccionar un cliente de una ruta (a)
- Insertar un cliente en una ruta (b)
- Intercambiar dos clientes (c)
- Crear una nueva ruta (p)

En el VRP con flota heterogénea y finita (VRPHE) se emplean las operaciones mencionadas anteriormente, y tres operaciones más que están relacionadas con los vehículos que deben visitar a los clientes en cada ruta:

- Seleccionar el vehículo de una ruta (v)
- Seleccionar un vehículo de la lista de vehículos disponibles (w)
- Intercambiar dos vehículos $([v, v])$

En este problema la flota de vehículos es finita y puede ocurrir que existan vehículos que no están asignados a ninguna ruta. De modo que solo con el conjunto de rutas no es posible representar una solución para este problema. Es necesario tener como parte de la solución el conjunto de los

vehículos que no han sido asignados a ninguna ruta. La operación *Seleccionar un vehículo de la lista de vehículos disponibles* selecciona un vehículo del conjunto de vehículos que no han sido asignados a ninguna ruta.

Existen otras variantes del VRP en las que también se dispone de elementos que no han sido asignados a ninguna ruta. Estos elementos pueden ser vehículos como en el problema anterior, clientes en problemas donde no es obligatorio visitar a todos los clientes, y depósitos en problemas donde existen múltiples depósitos. Existen operaciones que permiten seleccionar o insertar estos elementos. Ejemplos de este tipo de operaciones son *Seleccionar un vehículo de la lista de vehículos disponibles*, *Seleccionar un cliente de la lista de clientes no visitados* y *Seleccionar un depósito de la lista de depósitos disponibles*.

Existen operaciones que seleccionan o insertan elementos en una ruta. Estas operaciones requieren que se haya seleccionado una ruta en una operación anterior. Algunas de estas operaciones requieren además que si en un mismo criterio de vecindad existe más de una ocurrencia de una determinada operación, la ruta que se selecciona para ambas ocurrencias sea diferente. Por ejemplo, para la operación seleccionar el vehículo de una ruta es necesario que las rutas sean difenerentes, porque no tiene sentido seleccionar dos o más veces el vehículo de una misma ruta.

Es por esto que una parte de la descripción de las operaciones básicas de un VRP está dedica a clasificar las operaciones en dos categorías: operaciones consumidoras de r y operaciones exclusivas. En la próxima sección se explican las condiciones que debe cumplir una operación para ser clasificada en cada categoría.

2.2. Clasificación de operaciones básicas

Como segundo paso de la descripción de las operaciones básicas, estas se clasifican en: operación consumidora de r y operación exclusiva.

Definición 3 *Una operación es consumidora de r si para ejecutarse requiere que se haya seleccionado una ruta en una operación anterior.*

Ejemplos de este tipo de operaciones son *seleccionar un cliente de una ruta* e *insertar un cliente en una ruta*. Antes de ejecutar ambas operaciones se debe ejecutar la operación seleccionar una ruta para después insertar o seleccionar el elemento correspondiente.

Definición 4 *Una operación es exclusiva si es consumidora de r y además cumple que para dos instancias cualesquiera de dicha operación en un criterio de vecindad, la selección de ruta que consumen dichas instancias deben ser distintas.*

Las operaciones *seleccionar el vehículo de una ruta* y *seleccionar el depósito de una ruta* son operación consumidoras de r, pues requieren que antes se haya seleccionado una ruta. Estas operaciones se clasifican también como operaciones exclusivas, ya que no tiene sentido seleccionar más de una vez el vehículo (o depósito) de una ruta, en un mismo criterio de vecindad. Por eso, cada operación que seleccione un vehículo (o depósito) debe hacerlo de una ruta diferente.

Si en un criterio de vecindad existen operaciones consumidoras de r esto nos indica que delante de todas ellas debe existir al menos una instancia de la operación *Seleccionar una ruta* (r). De lo contrario existirían operaciones consumidoras de r para las cuales no existe operación para consumir.

La clasificación en las categorías operación consumidora de r y operación exclusiva permite eliminar criterios de vecindad infactibles. A través de la clasificación en operación consumidora de r, es posible eliminar aquellos criterios de vecindad en los que existen operaciones consumidoras de r para las cuales no se ha seleccionado una ruta en una operación anterior.

Por ejemplo, dado que la operación *Seleccionar un cliente de una ruta* es una operación consumidora de r, es posible eliminar el criterio de vecindad *arac*. Este criterio es infactible ya que para la primera ocurrencia de la operación *Seleccionar un cliente de una ruta* (a) no existe una operación anterior que seleccione una ruta (r).

A través de la clasificación en operación exclusiva también se eliminan los criterios de vecindad en los cuales existen ocurrencias de una misma operaciones exclusivas que consumen la misma operación de *Seleccionar una ruta*. Por ejemplo, el criterio de vecindad $rvv[v,v]$ no es factible ya que se esta seleccionando dos veces el mismo vehículo de una ruta para luego intercambiarlos, cuando en realidad no hay nada que intercambiar por que los dos son el mismo vehículo. Dado que la operación *Seleccionar el vehículo de una ruta* (v) es exclusiva, cada una de las instancias de esta operación debe consumir una operación de *Seleccionar una ruta* (r) diferente. Si se añade una operación de *Seleccionar una ruta* (r) delante de la segunda operación de *Seleccionar el vehículo de una ruta* (r), se obtiene el criterio $(rvrv[v,v])$ que sí es factible.

Especificar qué operaciones básicas pueden usarse en los criterios de vecindad para un VRP determinado y clasificar estas operaciones en las categorías antes mencionadas, no es suficente para generar criterios de vecindad factibles. Es necesario conocer que para insertar un cliente en una ruta este debe haber sido seleccionado en una operación anterior. En este trabajo se propone representar estas dependencias mediante reglas. En la próxima sección se define este concepto.

2.3. Especificación de reglas

El último paso de la descripción de las operaciones básicas es especificar el conjunto de reglas que expresan las relaciones de dependencia entre las operaciones básicas. Una regla tiene la forma:

$$\textit{operación de inserción} \xleftarrow{cond} \textit{secuencia de operaciones}.$$

Esto significa que para realizar la operación de la inserción antes debe ejecutarse la secuencia de operaciones de la derecha.

Por ejemplo, la regla

$$b \xleftarrow{cond} a$$

expresa que para insertar un cliente en una ruta este debe haber sido seleccionado en una operación anterior. Con las reglas

$$[v, v] \xleftarrow{cond} vv \tag{2.1}$$

$$[v, v] \xleftarrow{cond} vw \tag{2.2}$$

se expresa que para intercambiar dos vehículos $([v, v])$, estos deben ser seleccionados de dos rutas (v), como indica la regla 2.1, o de una ruta y de la lista de vehículos disponibles (w), como indica la regla 2.2.

Luego para describir el conjunto de operaciones que pueden utilizarse en los criterios de vecindad se debe: especificar y clasificar el conjunto de operaciones básicas, y por último especificar las reglas que rigen las relaciones de dependencia entre las operaciones básicas.

A modo de ejemplo en la próxima sección se presenta la descripción de las operaciones básicas del problema con flota heterogénea y finita (VRP-HE).

2.4. Ejemplo

En esta sección se presenta la descripción de las operaciones básicas que se pueden emplear en los criterios de vecindad para el VRPHE. Esta descripción constas de tres partes: primero se especifica el conjunto de operaciones básicas, luego estas operaciones se clasifican en las categorías operación consumidora de r y operación exclusiva, y por último se especifica el conjunto de reglas que plantean las relaciones de dependencia entre las operaciones básicas. En el VRPHE la flota de vehículos es finita y heterogénea, todos los clientes deben ser visitados por un único vehículo y existe un único depósito central.

Paso 1: Especificación de operaciones básicas

Primero se especifican las posibles operaciones básicas que pueden utilizarse en los criterios de vecindad para el problema que se analiza. Para el VRPHE estas operaciones pudieran ser:

- Seleccionar ruta (r)
- Seleccionar cliente de una ruta (a)
- Insertar cliente de una ruta (b)
- Intercambiar dos clientes (c)
- Crear nueva ruta (p)
- Seleccionar el vehículo de una ruta (v)
- Seleccionar un vehículo de la lista de vehículos disponibles (w)
- Intercambiar dos vehículos $([v,v])$

Las operaciones r, a, v y w son operaciones de selección ya que seleccionan rutas, clientes y vehículos respectivamente. Las operaciones b, c y $[v,v]$ son operaciones de inserción, ya que las dos primeras insertan clientes y la última inserta vehículos. La operación p también es de inserción, ya que

la creación de una nueva ruta requiere que en ella se inserten el vehículo asignado y al menos un cliente.

Paso 2: Clasificación de operaciones básicas

El segundo paso de la decripción de las operaciones básicas es clasificar estas operaciones en operación consumidora de r y operación exclusiva. El conjunto de operaciones básicas que se especificó en el paso 1 se debe clasificar de la siguiente forma:

- Operaciones consumidoras de r: a, b, v
- Operaciones exclusivas: v

Las operaciones consumidoras de r son a, b y v ya que para seleccionar un cliente o el vehículo de una ruta se debe seleccionar dicha ruta en una operación anterior, lo mismo ocurre si se quiere insertar un cliente en una ruta. La operación v es además exclusiva ya que no tiene sentido seleccionar dos veces el vehículo de una misma ruta en un mismo criterio de vecindad.

Paso 3: Especificación de reglas

El tercer y último paso de la descripción de las operaciones básicas es especificar el conjunto de reglas que plantean las relaciones de dependencia entre las operaciones básicas que se especifican en el paso 1. A continuación se muestran estas las reglas para el VRPHE:

$$b \xleftarrow{cond} a \tag{2.3}$$

$$c \xleftarrow{cond} aa \tag{2.4}$$

$$p \xleftarrow{cond} aw \tag{2.5}$$

$$[v, v] \xleftarrow{cond} vv \tag{2.6}$$

$$[v, v] \xleftarrow{cond} vw \tag{2.7}$$

La regla 2.3 expresa que para insertar un cliente en una ruta este debe haber sido seleccionado en una operación anterior. Para intercambiar dos clientes estos deben haber sido seleccionados, como plantea la regla 2.4. Para crear una nueva ruta esta debe seleccionar un cliente y un vehículo de la lista de vehículos disponibles como indica la regla 2.5. La regla 2.6

plantean que para intercambiar dos vehículos es necesario que estos sean seleccionados de dos rutas. Para intercambiar dos vehículos también es posible seleccionar un vehículo de una ruta y el otro de la lista de vehículos disponibles como indica la regla 2.7.

Una vez caracterizadas las operaciones básicas que pueden utilizarse en los criterios de vecindad para un VRP es posible generar automáticamente la gramática para dicho problema. En el siguiente capítulo se presenta el proceso de generación de la gramática.

Capítulo 3

Generación de la gramática

En este capítulo se presenta como obtener la gramática que genera los infinitos criterios de vecindad, a partir de la descripción de las operaciones básicas. El proceso de generación de la gramática consta de tres pasos:

1. generación del conjunto de símbolos terminales,
2. generación del conjunto de símbolos no terminales, y
3. generación de las producciones.

El conjunto de símbolos terminales de la gramática está formado por los símbolos que representan a las operaciones básica. Estas operaciones se especifican como primer paso en la descripción de las operaciones básicas de un VRP. En dependencia de cómo hayan sido clasificadas las operaciones básicas se genera el conjunto de símbolos no terminales. Este proceso se explica en la sección 3.1. Por último se genera el conjunto de producciones teniendo en cuenta el conjunto de reglas que se especifican en la descripción de las operaciones básicas. Este proceso se explica en la sección 3.2.

3.1. Generación de símbolos no terminales

En la mayoría de los problemas de enrutamiento de vehículos existen operaciones consumidoras de r. Las operaciones consumidoras de r son aquellas que requieren que antes se haya seleccionado una ruta como plantea la sección 2.2. Un ejemplo de este tipo de operaciones es la operación *Seleccionar un cliente de una ruta* (a).

Existe un subconjunto de las operaciones consumidoras de r que se clasifican, además, como operaciones exclusivas. Una operación es exclusiva si para dos ocurrencias de una misma operación, estas requieren que la operación de seleccionar ruta (r) que consumen sea diferente. Ejemplos de estas operaciones son *Seleccionar el vehículo de una ruta* (v) y *Seleccionar el depósito de una ruta* (d).

Para satisfacer los requerimientos de las operaciones consumidoras de r y de las operaciones exclusivas se generan los diferentes tipos de símbolos de la gramática. Para garantizar que para toda operación consumidora de r existe la operación de selección de ruta requerida, se generan los símbolos S y S^0. Para satisfacer los requerimientos de las operaciones exclusivas se generan símbolos de la forma S^k y $S^{k_1,k_2,\ldots,k_n}_{o_1,o_2,\ldots,o_n}$. En los siguientes epígrafes se analizan dos casos: cuando en el problema no existen operaciones exclusivas y cuando en el problema existen operaciones exclusivas.

3.1.1. Problemas sin operaciones exclusivas

En los problemas donde no existen operaciones exclusivas el único requerimiento que debe cumplirse es que para toda operación consumidora de r exista al menos una ruta que haya sido seleccionada en una operación anterior. Con los símbolos S y S^0 se garantiza el cumplimiento de dicho requerimiento.

El símbolo S^0 significa que en la forma oracional en la que él aparece ya existe al menos una operación de seleccionar ruta (r) que puede ser consumida por todas las operaciones consumidoras de r. Esto permite que en la parte derecha de todas las producciones donde el símbolo S^0 sea la parte izquierda, pueden aparecer operaciones consumidoras de r sin que esté presente una operación de selección de ruta (r).

Por ejemplo:

$$S^0 \rightarrow aaN_2c \tag{3.1}$$

$$S^0 \rightarrow araN_2c \tag{3.2}$$

en la producción 3.1 no existe ninguna operación de seleccionar ruta (r). En la producción 3.2 la primera operación consumidora de r (a) no tiene delante una operación de seleccionar ruta (r). Ambas producciones tienen sentido ya que el símbolo S^0 siginfica que ya ha sido añadida al menos una

operación de seleccionar ruta (r) en las formas oracionales en las que este símbolo puede aparecer.

El símbolo S es el símbolo inicial de la gramática. Este símbolo significa que en la forma oracional en la que él aparece no existen aún operaciones de seleccionar ruta. El símbolo S garantiza que exista una primera r delante de todas las operaciones consumidoras de r de la cadena. Esto se logra comprobando que cuando el símbolo S es la parte izquierda de una producción en cuya parte derecha existe al menos una operación consumidora de r, en la parte derecha de la misma también exista al menos una operación de selección de ruta (r). La operación de seleccionar ruta debe estar delante de la primera operación consumidora de r.

Ejemplos de estas producciones son,

$$S \rightarrow raaN_2c \tag{3.3}$$

$$S \rightarrow raraN_2c \tag{3.4}$$

En ambos casos la parte derecha de la producción incia con el símbolo r por lo que todas las operaciones consumidoras de r de la cadena van a tener delante la operación de seleccionar ruta que necesitan. En la producción 3.4 delante de la segunda operación consumidora de r (a) existe también un símbolo r.

Además de las operaciones consumidoras de r, en un problema también pueden existir operaciones exclusivas. Según la cantidad de operaciones de este tipo que existan en el problema, se generan símbolos no terminales de la forma S^k y $S^{k_1,k_2,\ldots,k_n}_{op_1,op_2,\ldots,op_n}$. El significado que tienen dichos símbolos y la razón por la que son generados se presentan en el siguiente epígrafe.

3.1.2. Problemas con operaciones exclusivas

Si en un problema VRP existen operaciones exclusivas se generan símbolos de la forma S^k y $S^{k_1,k_2,\ldots,k_n}_{op_1,op_2,\ldots,op_n}$. Por ejemplo, S^1, S^2_v, $S^{1,2}_{v,d}$.

El símbolo S^k significa que en la forma oracional en la que él aparece ya existen k selecciones de ruta (r) que pueden ser consumidas por cualquiera de las operaciones exclusivas del problema. Este símbolo solo se genera si existe más de una operación exclusiva en el problema.

Por ejemplo, supongamos que en el problema existen dos operaciones exclusivas *Seleccionar el vehículo de una ruta* (v) y *Seleccionar el depósito de una*

ruta (d), y que se ha generado la siguiente forma oracional $raraS^2c$. El símbolo S^2 indica que las 2 selecciones de ruta (r) en la forma oracional pueden ser consumidas por todas las operaciones exclusivas del problema. En este caso, por v, por d o por ambas. Es posible entonces sustituir el símbolo S^2 por expresiones donde no existan selecciones de ruta (r) delante de dos selecciones de vehículo (v) $raravv[v,v]c$, por dos selecciones de depósitos (d) $raradd[d,d]c$, o por ambas $raravvdd[d,d][v,v]c$.

El símbolo $S^{k_1,k_2,...,k_n}_{op_1,op_2,...,op_n}$ significa que en la forma oracional en la que él aparece ya existen k_i selecciones de ruta (r) que pueden ser consumidas por la operación exclusiva o_i, donde $1 \leq i \leq n$. Es decir, la operación op_1 puede consumir hasta k_1 selecciones de ruta (r), op_2 puede consumir k_2 selecciones de ruta (r), ..., la operación op_n puede consumir k_n selecciones de ruta (r). En estos casos, si no se dispone de selecciones de ruta (r) para una operación determinada, entonces dicha operación no se incluye en el símbolo.

Por ejemplo, el símbolo $S^{1,2}_{v,d}$ significa que en la forma oracional en la que aparezca este símbolo existe una operación de selección de ruta que puede ser consumida por la operación v y dos operaciones de selección de ruta que pueden ser consumidas por operaciones d. Es posible entonces sustituir este símbolo por expresiones donde no existan selecciones de ruta (r) delante de una selección de vehículo $rvv[v,v]$, dos selecciones de depósitos $dd[d,d]$, o por ambas $rvvdd[d,d][v,v]$.

Debe cumplirse que $1 \leq k, k_1, k_2, ..., k_n \leq m$, donde m se define de la siguiente forma:

Definición 5 *Sea p^o_i el número de ocurrencias de la operación exclusiva o en la regla i. El máximo número de ocurrencias exclusivas es $max\, p^o_i$ y se denota por m.*

3.1.3. Generación de símbolos según la descripción

Para toda variante del VRP se genera el símbolo S, que es el símbolo inicial. Si en el problema existen operaciones exclusivas se generan, además, símbolos de la forma S^k y $S^{k_1,k_2,...,k_n}_{op_1,op_2,...,op_n}$. Si en el problema no existen operaciones exclusivas se genera, además, el símbolo S^0

La cantidad de símbolos de la forma S^k y $S^{k_1,k_2,...,k_n}_{op_1,op_2,...,op_n}$ que se generan depende de la cantidad de operaciones exclusivas del problema (e) y del número máximo de ocurrencias exclusivas (m). Para ello a cada operación

se le asigna un índice, de modo que el conjunto de operaciones puede ordenarse con respecto a dicho índice. Luego se generan todas las cadenas de e dígitos $(k_1, k_2, ..., k_e)$, $0 \leq k_i \leq m \ \forall i = \overline{1,e}$, donde todos los dígitos son iguales o existe al menos un dígito que es 0. Con cada cadena se forma un símbolo de la forma $S^{k_1,k_2,...,k_e}_{op_1,op_2,...,op_e}$, donde $k_1, k_2, ..., k_e$ es la cadena de dígitos y $op_1, op_2, ..., op_e$ es el conjunto ordenado de las operaciones. De cada uno de estos símbolos se eliminan aquellos índices para los cuales $k_i = 0$, excepto cuando todos los k_i son 0. En problemas donde exite más de una operación exclusiva, los símbolos donde todos los k_i son iguales se transforman en símbolos S^k.

Por ejemplo, sea un problema en el que existe una única operación exclusiva v y $m = 2$. En este caso no es necesario definir un orden de operaciones ya que existe solo una operación exclusiva. Las cadenas de dígitos que se generan son 0, 1 y 2, obteniéndose los símbolos S^0, S^1_v y S^2_v. Además de estos símbolos se genera el símbolo inicial de la gramática S.

Sea un problema en el que existen dos operaciones exclusivas v y d. A la operación v se le asiga el índice 0 y a la operación d el índice 1. Las cadenas de dígitosque se generan son 00, 01, 02, 10, 11, 20 y 22, obteniéndose los símbolos $S^{0,0}_{v,d}$, $S^{0,1}_{v,d}$, $S^{0,2}_{v,d}$, $S^{1,0}_{v,d}$, $S^{1,1}_{v,d}$,$S^{2,0}_{v,d}$ y $S^{2,2}_{v,d}$. Se eliminan los índices donde $k_i = 0$: $S^{0,0}_{v,d}$, S^1_d, S^2_d, S^1_v, $S^{1,1}_{v,d}$, S^2_v y S^2. Dado que exite más de una operación exclusiva los símbolos donde todos los k_i son iguales se transforman en símbolos S^k: $S^{0,0}_{v,d} \Rightarrow S^0$, $S^{1,1}_{v,d} \Rightarrow S^1$ y $S^{2,2}_{v,d} \Rightarrow S^2$. Finalmente, se adiciona el símbolo incial S. Luego el conjunto de símbolos no terminales es: $\{S, S^0, S^1_d, S^2_d, S^1_v, S^1, S^2_v, S^2\}$.

Si en el problema no existen operaciones exclusivas el conjunto de símbolos terminales es: $\{S, S^0\}$.

Una vez generados los conjuntos de símbolos terminales y no terminales se pasa a la generación de las producciones. Este proceso se explica en la siguiente sección.

3.2. Generación de producciones

Al iniciar el proceso de generación de producciones de la gramática, ya se han generado los símbolos terminales y no termminales de la misma. Solo queda entonces generar el conjunto de producciones. Estas producciones

tienen todas la forma que se presenta en la figura 3.5:

$$N_1 \rightarrow o_1 o_2 ... o_n N_2 o, \tag{3.5}$$

N_1 y N_2 son símbolos no terminales de la forma S, S^k y $S^{k_1,k_2,...,k_n}_{op_1,op_2,...,op_n}$, $o_1 o_2 ... o_n$ es una secuencia de operaciones básicas de selección y o es una operación de inserción. Las producciones se obtienen a partir de las reglas que se especifican en la descripción de las operaciones básicas del problema. Estas reglas tienen la forma:

$$o \xleftarrow{cond} o_1 o_2 ... o_n. \tag{3.6}$$

donde $o_1, o_2, ..., o_n$ es una secuencia de operaciones básicas y o es una operación de inserción.

El proceso de generación de las producciones de la gramática consta de tres pasos:

- generación de pares secuencia-operación $(o_1 o_2 ... o_n, o)$,
- selección del conjunto de símbolos N_1 compatibles para cada par secuencia-operación,
- y selección del N_2 según el símbolo N_1 y el par secuencia-operación.

A continuación, se explica cada uno de estos pasos.

3.2.1. Generación de pares secuencia-operación

Cada producción de la gramática tiene la forma 3.5 y se obtiene a partir de una regla de la forma 3.6. En este epígrafe se describe como se obtiene los pares secuencia-operación $(o_1 o_2, ..., o_n, o)$.

La secuencia de operaciones básicas $o_1 o_2 ... o_n$ se obtiene a partir de la parte derecha de las reglas, y la operación de inserción o a partir de la parte izquierda. Por ejemplo, en la regla $b \xleftarrow{cond} a$ la secuencia de operaciones es a y la operación de inserción es b. En la regla $c \xleftarrow{cond} aa$ la secuencia de operaciones es aa y la operación de inserción es c. De forma general, a partir de la de una regla de la forma $o \xleftarrow{cond} o_1 o_2 ... o_n$ la secuencia de operaciones básicas es $o_1 o_2 ... o_n$ y la operación de inserción es o.

Según las operaciones que aparezcan en la regla que se analiza se generan otros pares secuencia-operación. Se presentan dos casos: si en la regla existen operaciones consumidoras de r en la parte derecha y si existen operaciones consumidoras de r en la parte izquierda. A continuación se analiza cada uno de estos casos.

Reglas con operaciones consumidoras de r en la parte derecha

Si la regla tiene operaciones consumidoras de r en la parte derecha entonces se generan todas las posibles secuencias que pueden contener o no una r delante de cada una de las operaciones consumidoras.

Por ejemplo, dada la regla $o_1 \xleftarrow{cond} o_2$, donde o_2 es una operación consumidora de r, las secuencias que se deben generar son o_2 y ro_2. De esta forma se obtienen los pares (o_2, o_1) y (ro_2, o_1).

Del mismo modo, si se analiza la regla $o_1 \xleftarrow{cond} o_2o_3$, donde o_2 y o_3 son operaciones consumidoras de r, entonces los pares serían (o_2o_3, o_1), (ro_2o_3, o_1), (o_2ro_3, o_1) y (ro_2ro_3, o_1).

Reglas con operaciones consumidoras de r en la parte izquierda

Si la operación de la parte izquierda de la regla es consumidora de r entonces por cada par ya generado en el paso anterior, se añade otro donde la secuencia de operaciones tiene una r al final.

Por ejemplo, a partir de la regla $o_1 \xleftarrow{cond} o_2$, donde o_1 es una operación consumidora de r. Si o_2 no es operación consumidora de r en el paso anterior se generó el par (o_2, o_1). En este nuevo paso se debe agregar el par (o_2r, o_1). Si o_2 es una operación consumidora de r en el paso anterior se generaron los pares (o_2, o_1) y (ro_2, o_1). En este nuevo paso se deben agregar los pares (o_2r, o_1) y (ro_2r, o_1).

Al aplicarle estos dos pasos a la regla $b \xleftarrow{cond} a$, donde a y b son operaciones consumidoras de r, se obtienen los siguientes pares secuencia-operación: (a, b), (ar, b), (ra, b) y (rar, b).

Al aplicarle estos pasos a la regla $c \xleftarrow{cond} aa$, donde a es una operación consumidora de r, se obtiene los siguientes pares secuencia-operación: (aa, c), (ara, c), (raa, c) y $(rara, c)$.

Las producciones de la gramática tienen la forma $N_1 \rightarrow o_1 o_2 ... o_n N_2 o$. Al finalizar este paso se tiene un conjunto de pares secuencia-operación $(o_1 o_2 ... o_n, o)$. Para completar la producción es necesario determinar cuáles no terminales se pueden insertar en esa producción. Ese análisis se presenta en las siguientes sección.

3.2.2. Selección del conjunto de símbolos N_1

Como segundo paso del proceso de generación de las producciones de la gramática se determina el conjunto de símbolos no terminales que pueden ser parte izquierda de las producciones. Para ello por cada par secuencia-operación se seleccionan los símbolos no terminales de la gramática que sean compatibles con él. Este símbolo tomará el lugar del no terminal N_1 en la producción de la forma 3.5.

Un par secuencia-operación y un símbolo no terminal son compatibles si el par cumple un conjunto de restricciones que dependen del símbolo no terminal. En este epígrafe se presenta, para cada tipo de símbolo no terminal (ver sección 3.1), las restricciones que debe cumplir un par secuencia-operación para que sean compatibles.

Restricciones de compatibilidad con el símbolo *S*

El símbolo *S* significa que en la forma oracional en la que él aparece no existen aún operaciones de seleccionar ruta. Este símbolo debe garantizar que exista al menos una operación de seleccionar ruta (r) delante de todas las operaciones consumidoras de r de la cadena.

Para ser compatible con el símbolo *S* los pares secuencia-operación deben cumplir que justo delante de la primera operación consumidora de *r* existe una operación de seleccionar ruta(r). Si en el par existen operaciones exclusivas, entonces delante de cada una de ellas también debe existir operación de seleccionar ruta (r).

Por ejemplos, los pares (ra, b) y $(rvrv, [v, v])$ son compatibles con *S*. En el primer caso existen dos operaciones consumidoras de *r*: *a* y *b*, y delante de *a*, que es la primera operación consumidora de *r*, existe una operación de seleccionar ruta (r). El segundo caso también es compatible porque delante de cada operación exclusiva del tipo seleccionar el vehículo de una ruta (v) existe una operación de seleccionar ruta (r).

El par (a, b) no es compatibles con el símbolo S, ya que no existe una operación de seleccionar ruta delante de la primera operaciones consumidora de r (a). Otro par que tampoco es compatible con S es $(rvv, [v, v])$, pues delante de la segunda operación exclusiva v no existe una operación de seleccionar ruta (r).

Restricciones de compatibilidad con el símbolo S^0

El símbolo S^0 significa que en la forma oracional en la que él aparece ya existe al menos una operación de seleccionar ruta (r) que puede ser consumida por todas las operaciones consumidoras de r. Esto permite que en la parte derecha de todas las producciones donde el símbolo S^0 sea la parte izquierda, pueden aparecer operaciones consumidoras de r sin que esté presente una operación de selección de ruta (r).

Los pares compatibles con el símbolo S^0 deben cumplir que si existen operaciones exclusivas, entonces delante de cada una debe existir una operación de seleccionar ruta (r). Cualquier otro par en el que no aparezcan operaciones exclusivas es compatible con S^0.

Por ejemplo, los pares (a, b) y $(rdrd, [d, d])$ son compatibles con S^0. En el primer caso no existen operaciones exclusivas y en el segundo caso, delante de cada operación exclusiva d existe una operación de seleccionar ruta (r).

Ejemplos de pares no compatibles con S^0 son $(rdd, [d, d])$ y $(dd, [d, d])$. En ambos ejemplos existe al menos una operación exclusiva (d) que no tiene delante una operación de seleccionar ruta (r).

Restricciones de compatibilidad con los símbolos S^k

El símbolo S^k significa que en la forma oracional donde él aparece existen k selecciones de ruta (r) que pueden ser consumidas por cualquiera de las operaciones exclusivas del problema. Para que un par secuencia-operación sea compatible con símbolos de la forma S^k el par debe cumplir ciertas restricciones que dependen de si en el par aparecen operaciones exclusivas o no.

En caso de existir operaciones exclusivas en el par, para ser compatible con el símbolo S^k el par debe cumplir que para toda operación exclusiva consume una operación de seleccionar ruta (r) diferente. Además debe cumplir una de las siguientes restricciones:

1. Para algún tipo de operación exclusiva debe cumplirse que el número de ocurrencias de esa operación, que no tienen delante una operación de seleccionar ruta (r), es igual a k.

2. Debe existir el mismo número de ocurrencias, que no tienen una operación de seleccionar ruta (r) delante, para todos los tipos de operaciones exclusivas del problema. Este número debe ser menor que k.

Cuando en el par no existen operaciones exclusivas para que este sea compatible con símbolos de la forma S^k debe cumplir que:

$$1 \leq count_r \leq m - k, \tag{3.7}$$

donde $(count_r)$ es la cantidad de operaciones de seleccionar ruta que aparecen en el par y m es el máximo número de ocurrencias exclusivas. Nótese que para los símbolos donde $k = m$, o sea S^m, no puede cumplirse la ecuación 3.7 ya que, $1 \leq count_r \leq 0$. Por lo tanto, solo se aceptaran pares donde existan operaciones exclusivas.

Supongamos que en el problema existen dos operaciones exclusivas v y d. Sea el símbolo S^1 y los pares $(rvv, [v, v])$ y $(drd, [d, d])$. En ambos casos delante de una de las operaciones exclusivas no existe una operación de seleccionar ruta (r), estas operaciones exclusivas consumen la operación de seleccionar ruta disponible en la forma oracional según el símbolo S^1, por lo que se cumple la restricción 1. Además se cumple que toda operación exclusiva consume una operación de seleccionar ruta diferente.

El par $(rvrv, [v, v])$ no es compatible con el símbolo S^1 ya que no cumplen la restricción 1. En este par todas las operaciones exclusivas tienen delante una operación de seleccionar ruta, por lo que ninguna operación exclusiva consume la operación de seleccionar ruta disponible en la forma oracional según el símbolo S^1. Para que el par $(rvrv, [v, v])$ cumpla la restricción 1 una de las operaciones exclusivas no debería tener delante una operación de seleccionar ruta (r).

Considérese un problema con dos operaciones exclusivas: (d_i) y (d_f), el símbolo S^2 y el par $(d_i d_f, [d, d])$. En este caso la restricción 1 no se cumple ya que deberían existir dos ocurrencias de una misma operación exclusiva que no tuviera delante una operación de seleccionar ruta. Sin embargo se cumple la restricción 2, ya que existe una ocurrencia de cada tipo de operación exclusiva del problema (d_i, d_f) que no tiene delante una operación de seleccionar ruta. Estas operaciones consumen la misma cantidad de selecciones

de rutas, y este número es menor que 2.

Supongamos que en el problema existen dos operaciones exclusivas, que $m = 2$ y sea el símbolo S^1. Pares compatibles con dicho símbolo son (ra, b) y (ara, c). En ambos casos no existen operaciones exclusivas, aparece una operación de seleccionar ruta (r), es decir $count_r = 1$, y se cumple la ecuación 3.7 ya que $1 \leq 1 \leq 2 - 1$.

Pares no compatibles serían (a, b) y (aa, c), ya que al no existir operaciones exclusivas debería aparecer al menos una operación de seleccionar ruta (r). Los pares (rar, b) y $(rara, c)$ tampoco son compatibles ya que no se cumple la ecuación 3.7 pues $1 \leq 2 \nleq 2 - 1$.

Restricciones de compatibilidad con los símbolos $S^{k_1,k_2,\ldots k_n}_{o_1,o_2,\ldots,o_n}$

El símbolo $S^{k_1,k_2,\ldots k_n}_{o_1,o_2,\ldots,o_n}$ significa que en la forma oracional en la que él aparece existen k_i selecciones de ruta (r) que pueden ser consumidas por la operación o_i, $1 \leq i \leq n$. Para que un par secuencia-operación sea compatible con símbolos de la forma $S^{k_1,k_2,\ldots k_n}_{o_1,o_2,\ldots,o_n}$ el par debe cumplir ciertas restricciones que dependen de si en el par aparecen operaciones exclusivas o no.

En caso de existir operaciones exclusivas en el par, para ser compatible con el símbolo $S^{k_1,k_2,\ldots k_n}_{o_1,o_2,\ldots,o_n}$ el par debe cumplir que toda operación exclusiva consume una operación de seleccionar ruta (r) diferente. Además debe cumplirse una de las siguientes dos restricciones:

1. Existe i, $1 \leq i \leq n$, tal que hay k_i ocurrencias de la operación o_i, que no tienen delante una operación de seleccionar ruta (r).
2. Existe i, $1 \leq i \leq n$ tal que toda ocurrencia de la operación o_i no tiene delante una operación de seleccionar ruta (r).

Cuando en el par no existen operaciones exclusivas para que este sea compatible con símbolos de la forma $S^{k_1,k_2,\ldots k_n}_{o_1,o_2,\ldots,o_n}$ debe existir solo una operación exclusiva en el problema. En el par debe existir al menos una operación de seleccionar ruta (r). Debe cumplirse también que la cantidad de rutas seleccionadas (r) en el par $(count_r)$ es menor que $m - k$, como se muestra en la ecuación 3.7. Nótese que para los símbolos donde $k = m$, o sea S^m_o, solo se aceptan pares donde aparezcan operaciones exclusivas ya que, no podría cumplirse las dos condiciones mencionadas anteriormente.

Sea el símbolo $S^{1,2}_{d,v}$, ejemplos de pares compatibles son $(rdd,[d,d])$ y $(vw,[v,v])$. En el primer caso existen dos operaciones exclusivas y para cada una de ellas existe una operación de seleccionar ruta (r) diferente: una de las ocurrencias de la operación d tiene delante una operación de seleccionar ruta y la otra, al no tener una operación de seleccionar ruta delante, consume la que está disponible en la forma oracional según el símbolo $S^{1,2}_{d,v}$, cumpliéndose la restricción 1. En el segundo caso se dispone de dos selecciones de ruta (r) para la operación exclusiva (v), y como para toda ocurrencias de dicha operación no exista delante una operación de seleccionar ruta dicho par es compatible, de modo que se cumple la restricción 2.

Sea $m = 2$ y v la única operación exclusiva del problema, entonces es posible que existan pares compatibles con el símbolo S^1_v donde no aparecen operaciones exclusivas. Ejemplos de dichos pares son (ra,b) y (raa,c). En ambos casos en la secuencia de operaciones existe una operación de seleccionar ruta (r), es decir $count_r = 1$ y se cumple la ecuación 3.7 pues $1 \leq 1 \leq 2-1$.

Pares no compatibles serían (rar,b) y (aa,c). En el primer caso no se cumple la ecuación 3.7, ya que $1 \leq 2 \nleq 2-1$. En el segundo caso, en el par no aparecen operaciones de seleccionar ruta (r).

Un mismo par puede ser compatible con más de un símbolo. Por ejemplo, el par (ra,b) es compatible con los símbolos S y S^0. Si en el problema existen varias operaciones exclusivas entonces sería compatible también con el símbolo S^1. Y si $m > 2$ sería compatible con S^2.

Para cada pareja $(N_1,(o_1o_2...o_n,o))$ de símbolo y par compatibles se construye una producción de la forma $N_1 \rightarrow o_1o_2...o_nN_2o$. Del par se obtienen la secuencia de operaciones $o_1o_2...o_n$ y la operación de inserción o, y el símbolo ocuparía el lugar del no terminal N_1. Para completar la producción solo faltaría seleccionar el no terminal N_2. En la proxima sección se explica cómo se debe seleccionar dicho símbolo.

3.2.3. Selección del N_2

Dado un par secuencia-operación y un símbolo N_1 se debe elegir un no terminal N_2 que complete la producción de la forma $N_1 \rightarrow o_1o_2...o_nN_2o$. Esta elección depende de las operaciones básicas que esten presentes en el problema.

Si en el problema no existen operaciones exclusivas, entonces para to-

da combinación de par secuencia-operación y símbolo N_1 se selecciona el símbolo S^0. Este símbolo se genera para todos los problemas VRP como se planteó en la sección 3.1.

En el CVRP no existen operaciones exclusivas, por lo que para cualquier combinación de par y símbolo debe seleccionarse el símbolo S^0 como símbolo N_2 para completar la producción. Por ejemplo para el par $(rara, c)$ y el símbolo S como N_1, se debe seleccionar como N_2 el símbolo S^0, obteniéndose la producción $S \rightarrow raraS^0c$.

Si en el problema existen operaciones exclusivas, se utiliza el símbolo N_1 y la secuencia de operaciones para determinar N_2. Para ello se calcula la cantidad de selecciones de ruta (r) disponibles en la forma oracional para cada operación exclusiva y se inserta el no terminal apropiado.

Para determinar ese no terminal se calcula para cada operación exclusiva la cantidad de selecciones de ruta (r) que puede consumir. Este número es la suma de las selecciones de ruta disponibles según el símbolo N_1 que no se emplearon, y la cantidad de selecciones de ruta (r) que aparecen en la secuencia de operaciones que pueden ser utilizadas para cada operación exclusiva del problema. Luego se determina el símbolo no terminal apropiado que se corresponde con dichas cantidades. Estos símbolos tienen la forma S^k o $S^{k_1,k_2,\ldots,k_n}_{op_1,op_2,\ldots,op_n}$.

En un problema con flota heterogénea y finita, en el que todos los clientes deben ser visitados una única vez, para el par $(rara, c)$ y el símbolo S como N_1, el símbolo seleccionado es S^2_v y la producción es $S \rightarrow raraS^2_vc$, dado que existe una sola operación exclusiva (v), se contaba con 0 selecciones de ruta (r), se añadieron 2 y no se consumió ninguna por la operación exclusiva v. Para el par $(rvrv, [v, v])$ y el símbolo S, el símbolo N_2 es S^0 y se obtiene la producción es $S \rightarrow rvrvS^0[v, v]$, en este caso se contaba con 0 selecciones de ruta (r) y se añadieron 2 que fueron consumidas por la operación exclusiva v.

Si en el problema existen las operaciones exclusivas v y d para el par $(rara, c)$ y el símbolo S como N_1, el símbolo seleccionado es S^2 y la producción es $S \rightarrow raraS^2c$, dado que existen dos operaciones exclusivas, se contaba con 0 selecciones de ruta (r) disponibles, se añadieron 2 y no se consumió ninguna. Para el par $(rvrv, [v, v])$ y el símbolo S, el símbolo N_2 debe ser S^2_d obteniéndose la producción $S \rightarrow rvrvS^2_d[v, v]$. Se contaba con 0 selecciones de ruta disponibles para ambas operaciones exclusivas, se añadieron 2 que fueron consumidas por la operación v por lo que aún pueden ser consumidas por la operación d.

Para el par $(rvv,[v,v])$ y el símbolo S^1 como N_1, el símbolo N_2 que se debe seleccionar es S^2_d y se obtiene la producción $S^1 \rightarrow rvvS^2_d[v,v]$. En este caso en la forma oracional ya existe una selección de ruta disponibles para ambas operaciones, como indica el símbolo S^1, esta operación se consume por la segunda operación v por lo que aún esta disponible para la operación d. Además en la secuencia de operaciones existe una operación de seleccionar ruta que se consume por la primera operación v por lo que también esta disponible para la operación d. Por lo tanto, se disponen de 0 rutas para la operación v y 2 para la operación d, y este es justamente el significado del símbolo S^2_d.

3.3. Ejemplo

En esta sección se muestra paso a paso como se obtiene la gramática que genera criterios de vecindad para el VRPHE a partir de la descripción de las operaciones básicas que pueden emplearse en los criterios de vecindad para dicho problema.

En el VRPHE la flota de vehículos es finita y heterogénea. Existe un único depósito y todos los clientes deben ser visitados una única vez. La descripción de las operaciones básicas para este problema se muestra a continuación.

Las operaciones básicas que pueden emplearse son:

- Seleccionar ruta (r)
- Seleccionar cliente de una ruta (a)
- Insertar cliente de una ruta (b)
- Intercambiar dos clientes (c)
- Crear nueva ruta (p)
- Seleccionar el vehículo de una ruta (v)
- Seleccionar un vehículo de la lista de vehículos disponibles (w)
- Intercambiar dos vehículos $([v,v])$

Las reglas que plantean las relaciones de dependecia entre estas operaciones son:

$$b \xleftarrow{cond} a$$
$$c \xleftarrow{cond} aa$$
$$p \xleftarrow{cond} aw$$
$$[v,v] \xleftarrow{cond} vv$$
$$[v,v] \xleftarrow{cond} vw$$

A continuación se explica el proceso de generación de la gramática para el VRPHE:

Paso 1: Generación de símbolos terminales

La primera parte de la descripción de las operaciones básicas de un VRP es la que especifica que operaciones pueden utilizarse en los criterios de vecindad en el problema. Esto nos permite formar el conjunto de símbolos terminales del mismo. Por tanto, el conjunto de símbolos terminales de la gramática para este problema es $r, a, b, c, p, v, w, [v,v]$.

Paso 2: Generación de símbolos no terminales

Como S y S^0 están presentes en la gramática de todos los VRP, estos símbolos pertenecen al conjunto de no terminales de este problema. El símbolo S es además el símbolo inicial de la gramática. Al existir solo una operación exclusiva (v) y al ser 2 el máximo número de ocurrencias exclusivas (m), se generan también los no terminales S_v^1 y S_v^2.

Entonces el conjunto de símbolos no terminales para este problema es:

$$S, S^0, S_v^1, S_v^2.$$

Seleccionados el conjunto de símbolos terminales y no terminales de la gramática, el siguiente paso es generar el conjunto de producciones.

Paso 3 Generación del conjunto de producciones

El primer paso del proceso de generación de las producciones de la gramática es generar las combinaciones secuencia-operación. Según las reglas planteadas anteriormente en la siguiente tabla se muestran las combinaciones secuencia-operación generadas para cada regla.

$b \xleftarrow{cond} a$	$c \xleftarrow{cond} aa$	$p \xleftarrow{cond} aw$	$[v,v] \xleftarrow{cond} vw$	$[v,v] \xleftarrow{cond} vv$
(a,b)	(aa,c)	(aw,p)	$(vw,[v,v])$	$(vv,[v,v])$
(ra,b)	(raa,c)	(raw,p)	$(rvw,[v,v])$	$(rvv,[v,v])$
(ar,b)	(ara,c)			$(vrv,[v,v])$
(rar,b)	$(rara,c)$			$(rvrv,[v,v])$

El segundo paso es seleccionar los símbolos que son compatibles con cada una de las combinaciones generadas. En la siguiente tabla se muestra las combinaciones compatibles para cada símbolo:

S	S^0	S^1_v	S^2_v
(ra,b)	(a,b)	(ra,b)	$(vw,[v,v])$
(rar,b)	(ra,b)	(raa,c)	$(vv,[v,v])$
(raa,c)	(ar,b)	(raw,p)	
$(rara,c)$	(rar,b)	$(vw,[v,v])$	
(raw,p)	(aa,c)	$(rvv,[v,v])$	
$(rvw,[v,v])$	(raa,c)	$(vrv,[v,v])$	
$(rvrv,[v,v])$	(ara,c)		
	$(rara,c)$		
	(aw,p)		
	(raw,p)		
	$(rvw,[v,v])$		
	$(rvrv,[v,v])$		

El tercer y último paso sería seleccionar el símbolo N_2 según el símbolo N_1 y la combinación secuencia-operación compatibles que completa una producción de la forma $N_1 \rightarrow o_1 o_2 ... o_n N_2 o$.

Con el par (ra,b) y el símbolo S se forma la producción $S \rightarrow raN_2b$. Según el símbolo S se cuenta con 0 selecciones de ruta (r) en la forma oracional. En el par existe una operación de seleccionar ruta (r) que puede ser consumida por la única operación exclusiva del problema. Sumando estas cantidades se tiene que existe una operación de seleccionar ruta que puede ser consumida por la única operación exclusiva del problema (v). Por tanto el símbolo N_2 que completa la producción es el símbolo S^1_v. Luego la producción que se forma es:

$$S \rightarrow raS^1_v b$$

Lo mismo ocurre para los pares (raa,c) y (raw,p) y el símbolo S. Luego se forman las producciones:

$$S \to raaS_v^1c$$
$$S \to rawS_v^1p$$

Para los pares (ra,b), (ar,b), (raa,c), (ara,c) y (raw,p) y el símbolo S^0, también ocurre lo mismo. Luego se generan las producciones:

$$S^0 \to raS_v^1b$$
$$S^0 \to arS_v^1b$$
$$S^0 \to raaS_v^1c$$
$$S^0 \to araS_v^1c$$
$$S^0 \to rawS_v^1p$$

Con el par (rar,b) y el símbolo S se forma la producción $S \to rarN_2b$. Según el símbolo S se cuenta con 0 selecciones de ruta (r) en la forma oracional. En el par existen dos selecciones de ruta (r) que pueden ser consumidas por la única operación exclusiva del problema. Sumando estas cantidades se tiene que existen dos operaciones de seleccionar ruta que pueden ser consumidas por la única operación exclusiva del problema (v). Por tanto el símbolo N_2 que completa la producción es el símbolo S_v^2. Luego la producción que se forma es:

$$S \to rarS_v^2b.$$

Lo mismo ocurre para el par $(rara,c)$ y el símbolo S. Luego la producción que se forma es $S \to raraS_v^2c$. Para los pares (rar,b) y $(rara,c)$ y el símbolo S^0, también ocurre lo mismo. Luego se generan las producciones:

$$S^0 \to rarS_v^2b$$
$$S^0 \to raraS_v^2c$$

Con el par $(rvw,[v,v])$ y el símbolo S se forma la producción $S \to rvwN_2[v,v]$. Según el símbolo S se cuenta con 0 selecciones de ruta (r) en la forma oracional. En el par existe una selección de ruta (r) que es consumida por la única operación exclusiva del problema. Sumando estas cantidades se tiene

que existen 0 operaciones de seleccionar ruta que puede ser consumida por la única operación exclusiva del problema (v). Por tanto el símbolo N_2 que completa la producción es el símbolo S^0. Luego la producción que se forma es:

$$S \rightarrow rvwS^0[v,v]$$

Lo mismo ocurre con el par $(rvw,[v,v])$ y el símbolo S^0. Luego la producción que se genera es $S^0 \rightarrow rvwS^0[v,v]$. De forma similar ocurre con el par $(rvrv,[v,v])$ y los símbolos S y S^0. En ambos casos se cuenta con 0 selecciones de ruta (r) segun los símbolo S y S^0. En el par existen dos selecciones de ruta (r) que son consumidas por la única operación exclusiva del problema (v). Sumando estas cantidades se tienen 0 selecciones de ruta que pueden ser consumidas por la operación (v). Por tanto el símbolo que completa las dos producciones es el símbolo S^0. Luego se generan las producciones:

$$S \rightarrow rvrvS^0[v,v]$$
$$S^0 \rightarrow rvrvS^0[v,v]$$

Con el par (a,b) y el símbolo S^0 se forma la producción $S^0 \rightarrow aN_2b$. Según el símbolo S se cuenta con 0 selecciones de ruta (r) en la forma oracional y en el par existen 0 selecciones de ruta (r). Sumando estas cantidades se tiene que existen 0 operaciones de seleccionar ruta que pueden ser consumida por la única operación exclusiva del problema (v). Por tanto el símbolo N_2 que completa la producción es el símbolo S^0. Luego la producción que se forma es:

$$S^0 \rightarrow aS^0b$$

Lo mismo ocurre con los pares (aa,c) y (aw,p) y el símbolo S^0. Luego las producciones que se generan son:

$$S^0 \rightarrow aaS^0c$$
$$S^0 \rightarrow awS^0p$$

Con el par (ra,b) y el símbolo S_v^1 se forma la producción $S_v^1 \rightarrow raN_2b$. Según el símbolo S_v^1 se cuenta con una selección de ruta (r) en la forma

oracional que puede ser consumida por la operación (v). En el par existe una selección de ruta (r) que puede ser consumida por la operación (v). Sumando estas cantidades se tiene que existen dos operaciones de seleccionar ruta que pueden ser consumida por la única operación exclusiva del problema (v). Por tanto el símbolo N_2 que completa la producción es el símbolo S_v^2. Luego la producción que se forma es:

$$S_v^1 \rightarrow raS_v^2b$$

Lo mismo ocurre con los pares (raa, c) y (raw, p) y el símbolo S_v^1. Luego las producciones que se generan son:

$$S_v^1 \rightarrow raaS_v^2c$$
$$S_v^1 \rightarrow rawS_v^2p$$

Con el par $(vw, [v, v])$ y el símbolo S_v^1 se forma la producción $S_v^1 \rightarrow vwN_2[v, v]$. Según el símbolo S_v^1 se cuenta con una selección de ruta (r) en la forma oracional que puede ser consumida por la operación (v). En el par existe una operación exclusiva (v) que no tiene delante una operación de seleccionar ruta (r), por lo que la consume de la que está disponible según el símbolo S_v^1. Sumando estas cantidades se tiene que existen 0 operaciones de seleccionar ruta que pueden ser consumida por la única operación exclusiva del problema (v). Por tanto el símbolo N_2 que completa la producción es el símbolo S^0. Luego la producción que se forma es:

$$S_v^1 \rightarrow vwS^0[v, v]$$

De forma similar ocurre con los pares $(rvv, [v, v])$ y $(vrv, [v, v])$ y el símbolo S_v^1. En ambos pares existen dos operaciones exclusivas (v), y solo una de ellas tiene un operación de seleccionar ruta delante. La otra operación exclusiva (v) de los pares consume la operación de seleccionar ruta (r) disponibles según el símbolo S_v^1. Sumando estas cantidades se tiene que existen 0 operaciones de seleccionar ruta que pueden ser consumidas por la única operación exclusiva del problema (v). Por tanto el símbolo N_2 que completa la producción es el símbolo S^0. Luego las producción que se forman son:

$$S_v^1 \rightarrow rvvS^0[v, v]$$
$$S_v^1 \rightarrow vrvS^0[v, v]$$

Con el par $(vw, [v, v])$ y el símbolo S_v^2 se forma la producción $S_v^2 \rightarrow vwN_2[v, v]$. Según el símbolo S_v^2 se cuenta con dos selecciones de ruta (r) en la forma oracional que puede ser consumida por la operación (v). En el par existe una operación exclusiva (v) que no tiene delante una operación de seleccionar ruta (r), por lo que la consume de la que está disponible según el símbolo S_v^1. Sumando estas cantidades se tiene que existe una operación de seleccionar ruta que pueden ser consumida por la única operación exclusiva del problema (v). Por tanto el símbolo N_2 que completa la producción es el símbolo S_v^1. Luego la producción que se forma es:

$$S_v^2 \rightarrow vwS_v^1[v, v]$$

De forma similar ocurre con el par $(vv, [v, v])$ y el símbolo S_v^2. En este caso existen dos operaciones exclusivas (v) que no tienen delante una operación de seleccionar ruta (r). Cada operación exclusiva consume la operación de seleccionar ruta de las que estan disponibles según el símbolo S_v^2. Sumando estas cantidades se tiene que existen 0 operaciones de seleccionar ruta que pueden ser consumidas por la única operación exclusiva del problema (v). Por tanto el símbolo N_2 que completa la producción es el símbolo S^0. Luego la producción que se forma es:

$$S_v^2 \rightarrow vvS^0[v, v]$$

Finalmente a partir de la descripción de las operaciones básicas para el VRPHE que se presenta a continuación:

Paso 1 Especificación de operaciones básicas:

- seleccionar ruta (r)
- seleccionar cliente de una ruta (a)
- insertar cliente de una ruta (b)
- intercambiar dos clientes (c)
- crear nueva ruta (p)
- seleccionar el vehículo de una ruta (v)

- seleccionar un vehículo de la lista de vehículos disponibles (w)
- intercambiar dos vehículos $([v,v])$.

Paso 2 Clasificación de operaciones básicas:

- Operaciones consumidoras de r: r: a, b, v
- Operaciones exclusivas: v.

Paso 3 Especificación de reglas

$$b \xleftarrow{cond} a \tag{3.8}$$

$$c \xleftarrow{cond} aa \tag{3.9}$$

$$p \xleftarrow{cond} aw \tag{3.10}$$

$$[v,v] \xleftarrow{cond} vv \tag{3.11}$$

$$[v,v] \xleftarrow{cond} vw \tag{3.12}$$

se obtiene la siguiente gramática:

$S \to raS_v^1 b$
$S \to rarS^2 b_v$
$S \to raaS_v^1 c$
$S \to raraS_v^2 c$
$S \to raS_v^1 p$
$S \to rvrvS^0[v,v]$
$S \to rvwS^0[v,v]$

$S^0 \to aS^0 b$
$S^0 \to raS_v^1$b
$S^0 \to arS_v^1 b$
$S^0 \to rarS_v^2 b$
$S^0 \to aaS^0 c$
$S^0 \to raaS_v^1 c$
$S^0 \to araS_v^1 c$
$S^0 \to raraS_v^2 c$
$S^0 \to aS^0 p$
$S^0 \to raS_v^1 p$
$S^0 \to rvrvS^0[v,v]$
$S^0 \to rvwS^0[v,v]$
$S^0 \to \varepsilon$

$S_v^1 \to raS_v^2 b$
$S_v^1 \to arS_v^2 b$
$S_v^1 \to raaS_v^2 c$
$S_v^1 \to araS_v^2 c$
$S_v^1 \to raS_v^2 p$
$S_v^1 \to vwS^0[v,v]$
$S_v^1 \to rvvS^0[v,v]$
$S_v^1 \to vrvS^0[v,v]$
$S_v^1 \to S^0$

$S_v^2 \to vwS_v^1[v,v]$
$S_v^2 \to vvS^0[v,v]$
$S_v^2 \to S_v^1$
$S_v^2 \to S^0$

En los capítulos 2 y 3 fueron detallados los procesos de descripción de las operaciones básicas que pueden emplearse en un VRP y como a partir de esa descripción se obtiene la gramática que genera infinitos criterios de vecindad para el problema que se analiza, respectivamente. En el próximo capítulo se muestran los detalles de implementación para cada uno de estos procesos.

Capítulo 4

Detalles de implementación

En este capítulo se presentan detalles de la implementación de la estrategia propuesta. En la primera sección se plantean los detalles referentes a la representación de la descripción de los problemas de enrutamiento de vehículos. En la segunda sección se exponen los detalles de la generación de la gramática.

4.1. Detalles de la representación de la descripción de los VRP

La descripción de un VRP propuesta en esta investigación consta de tres partes: especificación de operaciones básicas, clasificación de operaciones básicas y la especificación de las reglas que plantean la relación de dependencia entre las operaciones básicas. En esta sección se presentan los detalles de implementación que permiten expresar esta descripción en términos computacionales.

4.1.1. Jerarquía de operaciones

Para especificar las operaciones básicas que se utilizan y la clasificación de estas se propone la siguiente jerarquía de clases. La clase *operation* es la clase base de la jerarquía, que tiene como subclases a *select-operation*, *insert-operation* y *consumer-operation*. Las operaciones que hereden de las clases *select-operation* e *insert-operation* son operaciones de selección e inserción

respectivamente; las que hereden de la clase *consumer-operation* son clasificadas como operaciones consumidoras de *r*. La clase *consumer-operation* tiene una subclases llamada *exclusive-consumer-operation*, las operaciones que herenden de esta clase son clasificadas como operaciones exclusivas. En la figura 4.1 se representa esta jerarquía gráficamente.

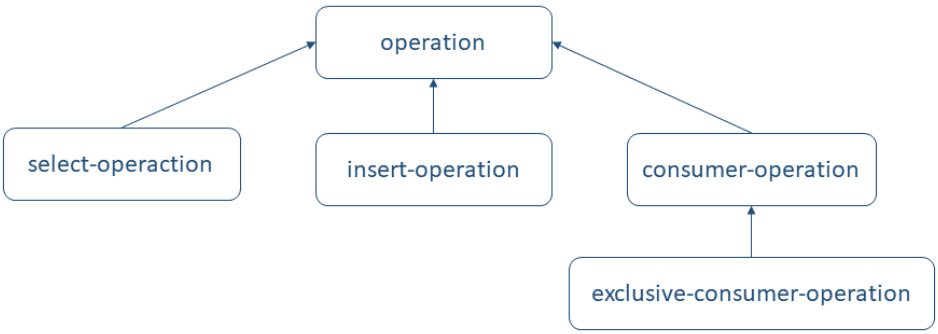

Figura 4.1: Jerarquía de operaciones

La clase operaciones define dos campos. El campo nombrado *terminal* contiene el símbolo terminal que representa a la operación. El otro campo de la clase se llama *description* y contiene una breve descripción de la operación.

Código 4.1: Definición de la clase *select-client*

```
1 (defclass select-client (select-operation consumer-operation)
2   ((terminal :initform "a")
3    (description :initform "Select a client from a route")))
```

En el fragmento de código 4.1 se define la operación seleccionar un cliente de una ruta a través de la clase *select-client*. Esta operación es de selección y se clasifica como operación consumidora de *r*. Estas dos características se expresan al heredar de las clases *select-operation* y *consumer-operation* respectivamente. El campo *terminal* plantea que el símbolo terminal que representa esta operación es la letra *a*.

Código 4.2: Definición de la clase *swap-clients*

```
1 (defclass swap-clients (insert-operation)
2   ((terminal :initform "c")
3    (description :initform "Swap two clients")))
```

La definición de la clase *swap-clients* (fragmento de código 4.2) representa la operación intercambiar dos clientes. Esta clase es una operación de inserción, esta características se expresa al heredar de la clase *insert-operation*. Esta operación se representa a través del símbolo terminal *c* como plantea la definición del campo *terminal*.

Código 4.3: Definición de la clase *select-vehicle*

```
1 (defclass select-vehicle (select-operation
2                           exclusive-consumer-operation)
3   ((terminal :initform "v")
4    (description :initform "Select a vehicle from a route")))
```

La clase *select-vehicle* representa la operación seleccionar el vehículo de una ruta, su definición se muestra en el fragmento de código 4.3. Esta operación es de selección como se indica al heredar de la clase *select-operation*, se clasifica como exclusiva al heredar de la clase *exclusive-consumer-operation*. También es una operación consumidora de *r* ya que la clase *exclusive-consumer-operation* hereda de la clase *consumer-operation*. El símbolo no terminal *v* representa a esta operación como se indica en la definición del campo *terminal*.

A través de esta jerarquía de clases se ejecutan los dos primeros pasos de la descripción de las operaciones básicas de un VRP: especificación de operaciones básicas y clasificación de operaciones básicas. El siguiente paso es definir la relaciones de dependencia entre las operaciones para el problema en cuestión. En la siguiente sección se plantean los detalles de implementación concernientes a esta parte de la descripción de un VRP.

4.1.2. Definición de componentes

En los problemas de enrutamiento de vehículos existen elementos fundamentales que están presentes en cada una de las variantes. Estos elementos son las rutas, los clientes, los vehículos y los depósitos. En dependencia de las características y restricciones que tengan estos elementos se definen las diferentes variantes del VRP.

Cuando se dispone de vehículos de diferentes tipos se dice que la flota de vehículos del problema es heterogénea, si todos los vehículos fueran iguales entonces la flota de vehículos es homogénea. Si existen varios depósitos entonces se dice que el problema tiene múltiples depósitos, por el contrario si existiese solo un depósito se dice que le problema presenta

un único depósito central. Si un cliente puede ser visitado por más de un vehículo entonces se dice que el problema es de demanda dividida.

En la presente investigación estas características y restricciones se representan a través de componentes. Una componente es una clase en la que se definen operaciones y reglas, las cuales expresan las posibles modificaciones que se pueden realizar en un criterio de vecindad respecto a una característica o restricción. Las operaciones y reglas se definen a través de métodos que pertenencen a las funciones genéricas *get-operations* y *get-rules* respectivamente.

La función *get-operations* recibe una instancia de una componente y devuelve las operaciones definidas para dicha componente. La función *get-rules* recibe una instancia de una operación de inserción y una instancia de una componente y devuelve las reglas definidas para la operación de inserción en la componente dada. Ambas funciones usan la combinación de operador *append* para devolver en una única lista los llamados a todos los métodos *get-rules* aplicables definidos para cada una de las clases de las que hereda la componente dada como parámetro.

Por ejemplo todos los problemas presentan rutas, esta característica se representa a través de la componente *there-are-routes*. Esta componente se define como se muestra en el fragmento de código 4.4:

Código 4.4: Definición de la componente *there-are-routes*

```
1 (defclass there-are-routes (commponent-VRP) ())
```

Las operaciones que esta componente requiere son:

- Seleccionar una ruta (r)
- Crear una nueva ruta (p)

Estas operaciones se obtienen a través de la definición del método *get-operations* para la componente *there-are-routes* como se aprecia en el fragmento de código 4.5.

Código 4.5: Definición método *get-operations* para la componente *there-are-routes*

```
2 (defmethod get-operations :append ((instance there-are-routes))
3   (list
4     (make-instance 'select-route)
5     (make-instance 'create-route)))
```

En esta componente no es necesario definir reglas.

Dado que las componentes son clases es posible crear una jerarquía de componentes. Como todos los problemas de enrutamiento de vehículos poseen rutas, todas las posibles componentes deben heredar de la componente *there-are-routes*. Esto permitirá utilizar las operaciones definidas en las componentes padre para la definicion de las reglas de una componente hija.

Por ejemplo la característica de que todos los clientes deben ser visitados una única vez se representa a través de la componente *client-partition*. Esta hereda de la componente *there-are-routes*. Luego las operaciones requeridas son:

- Seleccionar un cliente de una ruta (a)
- Insertar un cliente en una ruta (b)
- Intercambiar dos clientes (c).

El fragmento de código 4.6 muestra la defición de la componente *client-partition* y el método *get-operations* para esta componente.

Código 4.6: Definición de la componente *client-partition* y del método *get-operations* para la componente *client-partition*

```
1 (defclass client-partition (there-are-routes) ())
2
3 (defmethod get-operations :append (client-partition)
4   (list
5     (make-instance 'select-client)
6     (make-instance 'insert-client)
7     (make-instance 'swap-clients)))
```

Como la función genérica *get-operations* define la combinación de métodos de operador, no es necesario definir las operaciones seleccionar ruta y crear nueva ruta para la componente *client-partition*. El llamado al método *get-operations* con una instancia de esta componente devuelve una lista con el resultado de todos los métodos *get-operations* primarios aplicables, como se planteó en el epígrafe 1.4.2.

Como la componente *client-partition* hereda de la componente *there-are-routes*, y esta última define un método primario *get-operations*, el resultado del llamado es una lista con las 5 operaciones.

Las reglas que rigen las relaciones de dependecia entre las operaciones básicas de la componente *client-partition* son:

$$b \xleftarrow{cond} a$$
$$c \xleftarrow{cond} aa$$
$$p \xleftarrow{cond} a$$

La última regla expresa la relación de dependencia entre la operación p y a. La operación a esta definida en la nueva componente y la operación p está definida en la componente padre.

Estas reglas se definen a través de varios métodos *get-rules*. Por cada operación que aparezca en la parte izquierda de las reglas se crea una método distinto que devolverá la lista de las reglas en las que esta operación es la parte izquierda. A continuación se muestran estos métodos:

Código 4.7: Definición de métodos *get-rules* para la componente *client-partition*

```
8  (defmethod get-rules :append ((op insert-client)
9                                (instance client-partition))
10   (list
11     (make-instance 'rule
12                    :rigth (list
13                            (make-instance 'select-client))
14                    :left (make-instance 'insert-client))))
15
16 (defmethod get-rules :append ((op swap-clients)
17                                (instance client-partition))
18   (list
19     (make-instance 'rule
20                    :rigth (list
21                            (make-instance 'select-client)
22                            (make-instance 'select-client))
23                    :left (make-instance 'swap-clients))))
24
25 (defmethod get-rules :around ((op create-route)
26                                      (instance client-partition)
                                         )
27   (let ((rules (when
28                  (next-method-p)
29                    (call-next-method)))
30         (result nil))
31     (dolist (rule rules)
```

```
32      (dolist (rule2
33               (list
34                 (make-instance
35                   'rule
36                   :rigth (list
37                            (make-instance 'select-client))
38                   :left (make-instance 'create-route))))
39        (push (make-instance
40                'rule
41                :left (make-instance 'create-route)
42                :rigth (append
43                         (rule-rigth rule)
44                         (rule-rigth rule2)))
45              result)))
46    (if (null rules)
47        (list
48          (make-instance
49            'rule
50            :rigth (list (make-instance 'select-client))
51            :left (make-instance 'create-route)))
52        result)))
```

Los dos primeros método *get-rules* emplean la combinación de métodos a través del operador *append*. Estos métodos devuelven una lista que es el resultado de aplicar el operador *append* a los llamados de todos los métodos primarios aplicables en orden de más específico a menos específico como se plantean en el epígrafe 1.4.2. El tercer método es un método “en lugar de", el porqué se emplea este tipo de método para definir las reglas de la operación crear ruta se explica más adelante.

El código que se emplean para definir dos componentes diferentes es prácticamente el mismo. Solo el nombre de las operaciones que intervienen es diferente. Por esta razón se creó la macro *defcomponent*. La macro recibe el nombre de la nueva componente, la lista de componentes padres y las operaciones y reglas que la nueva componente define. Con estos datos se crea una nueva clase con el nombre de la componente, esta clase hereda de la lista de componentes padres, y definen los métodos *get-operations* y *get-rules* correspondientes. A continuación se muestra un ejemplo en el que se emplea la macro *defcomponent* para crear la componente *client-partition*.

Código 4.8: Definición de la componente *client-partition* a través de la macro *defcomponent*

```
1 (defcomponent client-partition (there-are-routes)
2   :new-operations
3     (insert-client swap-clients select-client)
```

```
4   :new-rules
5     ((insert-client <- select-client)
6     (swap-clients <- select-client select-client)
7     (create-route <-* select-client)))
```

Este llamado genera el siguiente código:

Código 4.9: Expanción del llamado a la macro *defcomponent* del fragmento de código 4.8

```
1  (progn
2    (defclass client-partition (there-are-routes) ())
3
4    (defmethod get-operations append ((instance client-partition)
       )
5      (list
6        (make-instance 'select-client)
7        (make-instance 'swap-clients)
8        (make-instance 'insert-client)))
9
10   (defmethod get-rules append ((op insert-client)
11                                      (instance client-partition
                                           ))
12     (list
13       (make-instance 'rule
14                      :rigth (list (make-instance 'select-client
                                    ))
15                      :left (make-instance 'insert-client))))
16
17   (defmethod get-rules append ((op swap-clients)
18                                (instance client-partition))
19     (list
20       (make-instance 'rule
21                      :rigth (list
22                                (make-instance 'select-client)
23                                (make-instance 'select-client))
24                      :left (make-instance 'swap-clients))))
25
26   (defmethod get-rules :around ((op create-route)
27                                 (instance client-partition))
28     (let ((rules (when
29                    (next-method-p)
30                      (call-next-method)))
31           (result nil))
32       (dolist (rule rules)
33         (dolist
34             (rule2 (list
35                       (make-instance
36                          'rule
```

```
37                          :rigth (list (make-instance '
                                select-client))
38                          :left (make-instance 'create-route))))
39          (push (make-instance 'rule
40                               :left (make-instance '
                                     create-route)
41                               :rigth (append (rule-rigth rule)
42                                              (rule-rigth rule2
                                                  )))
43                result)))
44      (if (null rules)
45        (list
46          (make-instance 'rule
47                         :rigth (list
48                                  (make-instance 'select-client
                                       ))
49                         :left (make-instance 'create-route)))
50        result))))
```

Como se puede observar se generan la misma clase y los mismo métodos *get-operations* que los definidos en el fragmento de código 4.7.

El tercer método *get-rules* es un método "en lugar de" y como indica la combinación de métodos estándar se llamará este en lugar del método primario. A través del llamado a la función *call-next-method* se ejecuta el siguiente método aplicable según la combinación de métodos estándar. Los métodos "en lugar de" se generan para las reglas definidas con el símbolo $\leftarrow *$ como se muestra en el fragmento de código 4.9.

Los métodos "en lugar de" combinan las reglas que se obtienen para una operación en las diferentes componentes de un VRP en una única regla. Por ejemplo para la operación *crear ruta* (p) y la componente *client-partition* se obtiene la regla $p \xleftarrow{cond} a$; si para esta misma operación y otra componente se define la regla $p \xleftarrow{cond} w$. Entonces el método "en lugar de" combina ambas reglas en esta $p \xleftarrow{cond} aw$.

Si para una componente se define mas de una regla con el símbolo $\leftarrow *$ para una misma operación entonces el método *"en lugar de"* devuelve más de una regla. Por ejemplo si para una componente y la operación p se definen las reglas $p \xleftarrow{cond} a$ y $p \xleftarrow{cond} a_{split}$, al combinarlas con la regla $p \xleftarrow{cond} w$, que se obtiene de otra componente, se obtienen las reglas $p \xleftarrow{cond} aw$ y $p \xleftarrow{cond} a_{split}w$.

Un problema VRP puede definirse a partir de las componentes que in-

tervienen en dicho problema. Por ejemplo, el CVRP se representa a través de la componente *client-partition*. Si la componente que describe que la flota de vehículos es heterogénea y finita fuese *heterogeneous-finite-fleet* entonces el VRPHE donde la flota es heterogénea y finita se respresenta a través de las componentes *client-partition* y *heterogeneous-finite-fleet*.

En esta sección se presentaron los detalles de implementación que permiten representar en términos computacionales la descripción de las operaciones básicas que pueden emplearse en los criterios de vecindad para un VRP determinado. En la próxima sección se presentan los detalles de implementación que permiten obtner de forma automática la gramática que genera infinitos criterios de vecindad.

4.2. Detalles de la generación de la gramática

El proceso de generación de la gramática consta de tres pasos: generación del conjunto de símbolos terminales, generación del conjunto de símbolos no terminales y generación del conjunto de producciones.

Como se explicó en el capítulo 3 el conjunto de símbolos terminales se obtiene a partir de la especificación de las operaciones básicas que pueden utilizarse en los criterios de vecindad. Dichas operaciones se especifican en la definición de las componentes a través del método *get-operations* como se plantea en el epígrafe 4.1.2. Al ejecutar el método *get-operations* a un problema VRP se obtiene el listado de las operaciones que pueden utilizarse en los criterios de vecindad para dicho problema. Dicho listado es además el conjunto de símbolos terminales de la gramática.

El conjunto de símbolos no terminales de la gramática se obtiene a partir del llamado a la función *get-no-terminals*. Esta función recibe como parámetro una instancia de una componente VRP. Esta función devuelve el listado de símbolos no terminales de la gramática para dicha componente siguiendo lo planteado en la sección 3.1.

Para generar el conjunto de producciones de la gramática existen tres pasos. El primero de estos pasos es la generación de las combinaciones secuencia-operación como plantea la sección 3.2. Para generar estas combinaciones se implementó una función recursiva que reproduce el procedimiento explicado en el epígrafe 3.2.1.

El segundo paso del proceso de generación de las producciones de la

gramática es seleccionar el conjunto de símbolos N_1 compatibles para cada par secuencia-operación. En el algoritmo 5 se muestra el seudocódigo correspondiente a este paso. A través del llamado a la función genérica *get-production* se determina si un símbolo no terminal y una combinación secuencia-operación son compatibles siguiendo las restricciones planteadas en la sección 3.2.2. En caso de ser compatibles devuelve la producción que estos generan.

```
1 foreach no-terminal-symbol in set-of-no-terminal-symbols do
2 |   foreach combination in set-of-combinations-sequence-operation do
3 |   |   get-production(combination, no-terminal-symbol)
4 |   end
5 end
```

Algorithm 1: Seudocódigo del paso 2 del proceso de generación de las producciones

La selección del símbolo no terminal N_2 que completa la producción se lleva a acabo a través de la función genérica *go-to-terminal*. Dicha función recibe como parámetros el símbolo no terminal N_1 y la combinación secuencia-operación. Siguiendo lo planteado en el epígrafe 3.2.3 se determina el símbolo no terminal que completa la producción.

Conclusiones

En esta investigación se propone una estrategia para, dada la descripción de un problema VRP, generar de forma automática una gramática que permite obtener infinitos criterios de vecindad. Las gramáticas obtenidas pueden ser utilizadas en algoritmos donde sea necesario el empleo de criterios de vecindad para solucionar problemas VRP, en particular para IVNS [11]. Con esta estrategia es posible reducir el tiempo empleado por los investigadores evitando la construcción de las gramáticas de forma manual.

Luego del estudio de las particularidades y características comunes de las principales variantes del problema VRP se propone una forma de describir los mismos en función de operaciones básicas. Esta descripción en la mayoría de los casos se puede obtener con un análisis básico del problema en cuestión, permitiendo, con la estrategia propuesta, obtener gramáticas de 33 producciones partiendo de 8 operaciones 5 reglas.

La implementación de la estrategia propuesta está basada en una jerarquía de componentes que representan características que pueden presentar los problemas VRP. Las componentes implementadas permite definir algunas de las principales variantes del VRP como son: restricciones de capacidad (CVRP), flota heterogénea (HVRP), múltiples depósitos (MDVRP) y demanda dividida (SDVRP), así como las combinaciones de estas variantes. Dicha implementación puede ser extendida con nuevas característica implementando las respectivas componentes e incorporándolas a la jerarquía. Esta capacidad de extensión permitirá el uso de la herramienta para variantes del VRP no concebidas de forma inicial y para otras que puedan surgir en el futuro.

Recomendaciones

Como recomendaciones para trabajos futuros se propone utilizar las gramáticas generadas en la solución de instancias de problemas reales. Comparar los resultados obtenidos con los resultados que se obtienen al utilizar criterios de vecindad seleccionados manualmente.

Se recomienda, además, extender la jerarquía de componentes propuesta, de modo que puedan representarse variantes del VRP no concebidas en esta investigación.

También se propone implementar una herramienta que permita visualizar las componentes existentes en la jerarquía y que facilite la creación de nuevas componentes.

Bibliografía

[1] Paolo Toth; Daniele Vigo. *The Vehicle Routing Problem (Monographs on Discrete Mathematics and Applications)*. Monographs on Discrete Mathematics and Applications. SIAM, 2001. (Citado en las páginas 1, 2, 5 y 6).

[2] Liong Choong Yeun, Wan Rosmania Ismail, Khairuddin Omar, and Mourad Zirour. Vehicle routing problem: models and solutions. *Journal of Quality Measurement and Analysis*, 4(1):205–218, 2008. (Citado en las páginas 1 y 5).

[3] Jorge Rodríguez Pérez. Caracterización, modelado y determinación de las rutas de la flota en una empresa de rendering. Master's thesis, 2012. (Citado en las páginas 1, 5 y 6).

[4] Gustavo Alfredo Bula, Caroline Prodhon, Fabio Augusto Gonzalez, H. Murat Afsar, and Nubia Velasco. Variable neighborhood search to solve the vehicle routing problem for hazardous materials transportation. *Journal of Hazardous Materials*, 324:472–480, feb 2017. (Citado en la página 1).

[5] Dalia Diaz Sistachs. Modelación matemática del problema de entrega y recogida simultánea con trasbordo y limitación de tiempo y estrategia mixta de búsqueda por vecindades. Master's thesis, Facultad de Matemática y Computación. Universidad de La Habana, La Habana, Cuba., 7 2018. (Citado en la página 1).

[6] David Pisinger and Stefan Ropke. A general heuristic for vehicle routing problems. *Computers & Operations Research*, 34(8):2403–2435, aug 2007. (Citado en la página 1).

[7] Dafne García de Armas. Estrategia por fases aplicada al problema de enrutamiento de vehículos con flota heterogénea, múltiples comparti-

mentos y demanda dividida. Master's thesis, Facultad de Matemática y Computación. Universidad de La Habana, La Habana, Cuba., 7 2017. (Citado en la página 1).

[8] Ingrid Morejón Gracía. El problema de enrutamiento de vehículos con recogida y entrega simultánea, 6 2014. (Citado en la página 1).

[9] Luis Alberto Bouza Alvarez. Un algoritmo memético para el problema de planificación de recorridos turísticos considerando restricciones de tiempo y energía., 6 2016. (Citado en la página 1).

[10] Alina Fernández Arias. *Modelos Y métodos para el Problema de enrutamiento de vehículos con recogida y entrega simultánea.* PhD thesis, Facultad de Matemática y Computación. Universidad de La Habana, La Habana, Cuba., 4 2017. (Citado en la página 1).

[11] Camila Pérez Mosquera. Primeras aproximaciones a la búsqueda de vecindad infinitamente variable, 6 2017. (Citado en las páginas 2, 7 y 56).

[12] J. H. Dantzig, G. B.; Ramser. The truck dispatching problem. *Management Science*, 6, 10 1959. (Citado en la página 4).

[13] Said Salhi, Arif Imran, and Niaz A. Wassan. The multi-depot vehicle routing problem with heterogeneous vehicle fleet: Formulation and a variable neighborhood search implementation. *Computers & Operations Research*, 52:315–325, dec 2014. (Citado en la página 5).

[14] M. Filipec, D. Skrlec, and S. Krajcar. Darwin meets computers: new approach to multiple depot capacitated vehicle routing problem. In *1997 IEEE International Conference on Systems, Man, and Cybernetics. Computational Cybernetics and Simulation*. IEEE. (Citado en la página 5).

[15] Gilbert Laporte and Yves Nobert. Exact algorithms for the vehicle routing problem. In *Surveys in Combinatorial Optimization*, pages 147–184. Elsevier, 1987. (Citado en la página 6).

[16] Roberto Baldacci, Aristide Mingozzi, and Roberto Roberti. Recent exact algorithms for solving the vehicle routing problem under capacity and time window constraints. *European Journal of Operational Research*, 218(1):1–6, apr 2012. (Citado en la página 6).

[17] Michel Gendreau, Jean-Yves Potvin, Olli Bräumlaysy, Geir Hasle, and Arne Løkketangen. Metaheuristics for the vehicle routing problem and its extensions: A categorized bibliography. In *Operations Research/Computer Science Interfaces*, pages 143–169. Springer US. (Citado en la página 6).

[18] El-Ghazali Talbi. *Metaheuristics*. John Wiley & Sons, Inc., jun 2009. (Citado en la página 6).

[19] Jeffrey D. Ullman John E. Hopcroft, Rajeev Motwani. *Introduction to automata theory, languages, and computation*. Addison-Wesley, 2nd ed edition, 2001. (Citado en la página 7).

[20] Peter Seibel. *Practical Common Lisp*. Apress, 2005. (Citado en la página 12).