Universidad de La Habana
Matemática y Computación

Análisis de vecindades en problemas de enrutamiento de vehículos con múltiples depósitos y flotas heterogéneas

Autor: **Gabriela Mijenes Carrera**

Tutores: **MSc. Fernando Rodríguez Flores**

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

Noviembre, 2021

A los Mijenes, los Migenes, y los Carrera-Balado, en especial a mi mamá y mi papá, por su apoyo incondicional; a mi querida María Rosa, porque sin su exigencia nunca hubiera llegado a ser quien soy; a mi "husband in code", Julito; y a mi tutor Fernando por tantos dolores de cabeza que le debo haber provocado.

Opinión del tutor

Desde hace varios años, en la Facultad de Matemática y Computación se está desarrollando una herramienta que permita resolver la mayor cantidad de Problemas de Enrutamiento de Vehículos (VRP) con el menor esfuerzo humano posible.

En el proceso de solución de un VRP, uno de los pasos que más esfuerzo consume y más trabajo requiere, es el diseño y la implementación de los criterios de vecindad. Gracias al esfuerzo y al trabajo de Gabriela, ya es posible usar el sistema existente para resolver problemas de enrutamiento de vehículos que incluyan operaciones relacionadas con los vehículos y los depósitos.

Para realizar este trabajo Gabriela tuvo que estudiar temas y tecnologías que no forma parte de su plan de estudio, pensar creativamente para resolver los problemas que aparecieron durante la tesis, y además, leer, comprender y modificar código ajeno, en un lenguaje "más ajeno todavía".

Por todo eso (y más) estoy convencido de que Gabriela reúne todos los requisitos (y más) para desempeñarse como una excelente graduada (y mucho más) de Ciencia de la Computación.

MSc. Fernando Raul Rodriguez Flores
Facultad de Matemática y Computación
Universidad de la Habana
Noviembre, 2021

Resumen

En un mundo computarizado y globalizado como el actual, la distribución de recursos y personas se convierte en una tarea primordial. La necesidad ambiental y económica del aprovechamiento inteligente de los recursos y la máxima de que el tiempo es dinero nos obligan a pensar en mejores formas de cumplir esta tarea. El uso de la computación en la esfera de la transportación ha demostrado brindar excelentes resultados, con el empleo de algoritmos que intentan resolver el históricamente conocido problema de enrutamiento de vehículos o VRP por sus siglas en inglés, así como cada una de sus variantes. Con este objetivo, se desarrolla un sistema complejo basado en el uso de metaheurísticas de búsqueda local, con el que se colabora en esta investigación para agregar funcionalidades que permitan resolver VRP con depósitos múltiples y VRP con flota heterogénea, a partir de la adición de operaciones que permitan crear criterios de vecindad que comprendan las características de los problemas en cuestión.

Abstract

In today's computerized and globalized world, the distribution of resources and people becomes a primary task. The environmental and economic need for the intelligent use of resources and the maxim that time is money force us to think of better ways to accomplish this task. The use of computing in the field of transportation has proven to provide excellent results, with the use of algorithms that try to solve the historically known vehicle routing problem, as well as each of its variants. With this objective, a complex system is developed based on the use of local search metaheuristics, with which it collaborates in this research to add functionalities that allow solving VRP with multiple depots and VRP with heterogeneous fleet, from the addition of operations that allow the creation of neighborhood criteria that use the characteristics of the problems in question.

Índice general

1. Preliminares **5**
1.1. VRP 5
1.2. Metaheurísticas de búsqueda local 7
1.3. Criterios de vecindad o estructuras de entorno 7

2. Sistema **10**
2.1. Búsqueda de Vecindad Infinitamente Variable 10
2.2. Cardinalidad de las operaciones 11
2.3. Árbol de Vecindad 14
2.4. Indexación 16
2.5. Generadores 19

3. VRP con depósito múltiple **23**
3.1. Solución 23
3.2. Operaciones 24
3.3. Cálculo de la cardinalidad 25
3.4. Árbol de vecindad 26
3.5. Indexación 27
3.6. Ejemplo 31

4. VRP con flota heterogénea **36**
4.1. Solución 36
4.2. Operaciones 37
4.3. Cálculo de la cardinalidad 38
4.4. Árbol de vecindad 40
4.5. Indexación 40
4.6. Ejemplos 42

Introducción

En los últimos dos años el mundo fue partícipe de una nueva revolución: El covid-19. La pandemia ha obligado a todos a permanecer en casa, cambiando la forma de estudiar, trabajar, comprar, vivir. En este sentido, las ventas a domicilio tomaron el papel protagónico brindando una alternativa para satisfacer a los clientes sin que estos tuvieran que salir de casa poniéndose en riesgo. Esto, unido a la ya existente necesidad de transportar materia prima, insumos, personas, hace de la transportación una de las ramas más importantes a optimizar en el presente siglo.

Los trabajos e investigaciones dedicadas a dar respuesta a este problema se engloban bajo el nombre de Problema de Enrutamiento de Vehículos o VRP por sus siglas en inglés. En su forma más sencilla, este problema está compuesto por un depósito central, un conjunto de vehículos idénticos y un conjunto de clientes. Su objetivo es visitar los clientes usando los vehículos que salen y regresan del depósito central de forma que se optimice cierto criterio: costo de transportación de los productos, tiempo de espera de los clientes, entre otros; pero realmente hay tantas variaciones del problema como necesidades puedan presentarse en la realidad [1][2].

La primera referencia a este tipo de problemas es el "Problema de despacho de camiones"para modelar la distribución de combustible a un grupo de gasolineras por camiones idénticos minimizando la distancia desde un depósito central [3].

Posteriormente, se define el hoy conocido problema clásico de enrutamiento de veículos, por Clark y Wright [4].

Con el paso del tiempo aparecen en la literatura problemas diferentes. Ahora no solo importan el orden en que se visita a los clientes o la asignación de estos a una ruta, sino que aparecieron también VRP con ganancias [5], donde se decide qué clientes visitar; o problemas donde hay varios depósitos y se debe escoger entonces desde donde comenzar el recorrido de cierta ruta; o los vehículos tienen características diferentes y en dependecia de esto pueden servir mejor(o peor) a los clientes.

Desde 2002 se han hecho estudios donde la aplicación de métodos computacionales han resultado en dismiuciones entre el 5 % y el 20 % de los gastos del costo de la transportación. El VRP es un problema NP-duro [6], por lo que los métodos exactos solo son factibles para problemas de pequeña dimensión, pero para problemas con más de 100 clientes se torna realmente complicado [7]. Para problemas reales con varios cientos de clientes, la vía

para encontrar las soluciones suelen ser las metaheurísticas.

Desde el año 2017, en la Facultad de Matemática y Computación de la Universidad de La Habana se trabaja en la elaboración de un sistema que permita obtener soluciones de diversos tipos de VRP, con el menor esfuerzo humano posible. En este año, se hacen las primeras aproximaciones a la Búsqueda de Vecindad Infinitamente Variable [8], en la que es posible considerar infinitos criteros de vecindad, y no solamente un número finito de estos como es usual. Esto elimina la necesidad de escoger qué criterios utilizar ya que se usarían todos los criterios posibles.

Para considerar infinitos criterios de vecindad se usan gramáticas libres del contexto que pueden ser diferentes para distintos VRP. En 2019 [9] se presenta una idea para generar de forma automática dichas gramáticas a partir de una descripción de las operaciones que se pueden aplicar a una solución de ese problema. Esta propuesta puede contribuir a reducir el tiempo empleado por los investigadores evitando la construcción manual de las mismas.

En 2020, se presenta un método que permite obtener el costo de soluciones vecinas de una solución dada en un VRP cualquiera, sin tener que implementar un algoritmo para ello [10]. Lo único que se necesita es el código de evaluación de una solución dada para la variante del VRP deseada. Esto es posible porque cuando se evalúa la solución se construye un grafo que contiene todas las operaciones usadas para el cálculo de la solución. Para obtener el costo de una solución vecina, solo es necesario modificar este grafo apropiadamente.

Además, en este mismo año, se presenta una propuesta para particionar una vecindad cualquiera en regiones y estimar las "mejores" regiones de la vecindad para intensificar la búsqueda en ellas [11]. Esta idea se implementa en un algoritmo de dos fases, en el que se usan generadores de soluciones vecinas como una implementación perezosa de las estrategias de exploración. Los resultados de este trabajo indican que el algoritmo de exploración en dos fases propuesto es significativamente superior a la estrategia aleatoria, siendo el enfoque adecuado si se desea aumentar el número de soluciones mejores analizadas durante la exploración de la vecindad.

Combinando estos resultados es posible disminuir el tiempo humano dedicado a la solución de un VRP. Con la propuesta [10] no es necesario implementar cómo evaluar las soluciones vecinas. Con [9] y [8] es posible considerar todos los posibles criterios de vecindad, y con las ideas presentadas en [11] es posible explorar estas vecindades "de la mejor manera posible". Sin embargo, el sistema propuesto tiene un inconveniente. Las ideas propuestas en

[11] solo se pueden aplicar en VRP, cuyos criterios de vecindad no incluyan operaciones con los depósitos o los vehículos de las rutas. Eso quiere decir que problemas con múltiples depósitos o flota heterogénea, no pueden ser resueltos con esta estrategia. En este trabajo se elimina esta restrición.

El objetivo de este trabajo es diseñar, implementar y agregar al sistema ya existente las operaciones necesarias para resolver variantes del VRP que incluyan múltiples depositos y flota heterogénea. Para ello se han trazado los siguientes objetivos específicos:

- Estudio del sistema existente al que se debe agregar la funcionalidad.
- Estudio de las variantes del problema que son objetivo a resolver en la presente investigación.
- Estudio, diseño e implementación de las operaciones involucradas en la solución de los problemas escogidos

En el primer capítulo de este documento se presentan los elementos fundamentales del VRP, así como el sistema al que se le añadirán las nuevas funcionalidades. En el segundo capítulo se profundiza en la solución existente para CVRP en el sistema al que tributa esta investigación. En los capítulos tercero y cuarto se presentan los problemas MDVRP y HFVRP respectivamente y se brinda el pseudocódigo de la solución propuesta, así como otros detalles de implementación.

Capítulo 1

Preliminares

En este capítulo se presenta la base teórica ya existente sobre la que se construye la presente investigación. En la sección 1.1 se presenta el Problema de Enrutamiento de Vehículos (VRP, por sus siglas en inglés) y las variantes sobre las que se estará centrando el trabajo. En las secciones 1.2 y 1.3 se presentan los conceptos y definiciones relacionados con metaheurísticas de búsqueda local y criterios de vecindad, como factores teóricos de gran relevancia para la solución presentada.

1.1. VRP

En el VRP se tiene conjuntos de clientes, depósitos y vehículos, cada uno con características específicas. En general se desea visitar a los clientes con vehículos que parten de los depósitos, optimizando algún criterio (que depende del problema específico) y asegurando que se satisfagan todas las restricciones, que pueden variar de un problema a otro.

Muchas son las variaciones del VRP que pueden ser citadas, pero se centrará la atención a las englobadas en el presente estudio: VRP con múltiples depósitos y VRP con flota heterogénea [2] [12].

La versión más simple del VRP es el Problema de Enrutamiento de Vehículos con restricciones de Capacidad (CVRP por sus siglas en inglés), que puede definirse formalmente como un conjunto de n clientes que deben ser servidos por m vehículos idénticos. Cada cliente c_i tiene una demanda r_i y la suma de las demandas de los clientes asignados a un vehículo k deben de ser menor que la capacidad de ese vehículo. Todos los vehículos deben

S = ((1 2 3)
(4 5)
(6))

Figura 1.1: Ejemplo de solución CVRP

comenzar y terminar su ruta en el depósito.

En estos casos, la solución S_1 del problema se puede representar como un conjunto de rutas r_i, estas a su vez están compuestas por listas de clientes, como se aprecia la figura 1.1.

Esta solución pertenece a una instancia del problema para seis clientes, donde se crearon tres rutas para brindar el servicio.

En la realidad los problemas se hacen mucho más complejos, por ejemplo, una compañía puede tener muchos depósitos para atender a sus clientes. Si los clientes se agrupan alrededor de los depósitos, entonces el problema de distribución debe ser modelado con un conjunto de VRP independientes. Sin embargo, si los clientes y los depósitos están entremezclados entonces se debe plantear como un problema VRP multidepósito (MDVRP, Multiple-Depot VRP) [12].

Si se centra la atención en los vehículos, pueden surgir problemas donde todos los componentes de la flota sean idénticos (como en el CVRP), pero en la vida real, la mayoría de las empresas cuentan con vehículos de distintas capacidades, algunos gastan más combustible, o son más lentos. En ocasiones, todas estas cualidades son importantes, y algunas hasta son inversamente proporcionales a otras (un vehículo que permita mayor carga puede ser más lento, o un vehículo más rápido puede consumir mayor cantidad de combustible) y entonces el problema se hace más complejo [12].

A esta familia de VRPs se les conoce como problemas de enrutamiento de vehículos de flota heterogénea, y estos a su vez se dividen en dos grupos: las flotas donde tenemos infinitos vehículos de cada clasificación (HFVRP, por sus siglas en inglés) y las flotas donde tenemos una cantidad fija de vehículos de cada clasificación (HFFVRP, por sus siglas en inglés)[12].

Como ya se ha mencionado en secciones anteriores, estos problemas se hacen incomputables para instancias grandes de los datos con los métodos de búsqueda de solución exacta. La solución propuesta en el sistema al que tributa este trabajo está basada en metaheurísticas de búsqueda local. Sobre este tema se profundiza en la siguiente sección.

1.2. Metaheurísticas de búsqueda local

Las metaheurísticas de búsqueda local parten de una solución inicial que se modifica, de acuerdo a un criterio de vecindad definido por el usuario. La solución modificada se acepta o se rechaza en dependencia del algoritmo en cuestión. Por ejemplo, en Recorrido Simulado [13] se aceptan soluciones peores de acuerdo a una probabilidad, finalmente, se devuelve la mejor solución obtenida durante el proceso. Una de las metaheurísticas de búsqueda local que ha sido aplicada con éxito al VRP es Búsqueda de Vecindad infinitamente variable.

Un factor importante dentro de estas metaheurísticas son los criterios de vecindad o estructuras de entorno. Así se le llama a las modificaciones aplicadas a una solución de VRP para generar un conjunto de nuevas soluciones, conocido como vecindad.

En la siguiente sección se profundiza en el tema de las estructuras de entorno.

1.3. Criterios de vecindad o estructuras de entorno

Los criterios de vecindad o estructuras de entorno son un factor clave en el diseño de algoritmos de búsqueda local ya que definen la forma en la que se modifica la solución inicial para generar nuevas soluciones. Formalmente, se define un criterio de vecindad en el espacio X de las soluciones del VRP como una aplicación $\eta : X \rightarrow 2^X$ tal que cada solución $x \epsilon X$ se le asigna un conjunto de soluciones η(x) llamadas soluciones vecinas de x, o vecindad de x.

Los criterios de vecindad que se usan dependen de cada problema, y de cómo se representen las soluciones de ese problema. En el caso del VRP clásico, en el que las soluciones se reprensentan mendiante listas de clientes, se pueden construir estructuras de entorno como las siguientes:

- Mover un cliente dentro de su ruta.
- Mover un cliente a otra ruta.
- Intercambiar dos clientes dentro de una misma ruta.

1. Seleccionar ruta r_1
2. Seleccionar cliente c_1
3. Seleccionar ruta r_2
4. Seleccionar cliente c_2
5. Intercambiar c_1 y c_2

Figura 1.2: Ejemplo de estructura de entorno

- Intercambiar dos clientes dentro de rutas diferentes.

Cada uno de estos criterios se pueden describir en términos de operaciones elementales como:

- Seleccionar una ruta.
- Seleccionar un cliente.
- Insertar un cliente dentro de una ruta.
- Intercambiar dos clientes previamente seleccionados.

Por ejemplo, la estructura de entorno : **Intercambiar dos clientes previamente seleccionados** se puede representar como en la figura 1.2.

Las operaciones elementales tienen un conjunto de propiedades cuyos valores determinan la solución vecina que se obtiene.

A continuación se muetra un ejemplo de las operaciones elementales para el CVRP y las propiedades que las definen:

- **Seleccionar ruta:** Esta operación se caracteriza por el número de la ruta seleccionada.
- **Seleccionar cliente:** Esta operación se caracteriza por la posición de la cual se selecciona cliente.
- **Insertar cliente:** Esta operación se caracteriza por la posición en la cual se inserta el cliente.

- **Intercambiar clientes:** se caracteriza por los dos clientes que deben ser intercambiados.

Estas operaciones básicas son la clave de la construcción de cualquier estructura de entorno para un VRP. Sin embargo, existen problemas en los que, para definir los criterios de vecindad es necesario definir nuevas operaciones. Por ejemplo, para el VRP con múltiples depósitos, es necesario definir operaciones que permitan cambiar los depósitos. En las secciones siguientes se profundiza sobre este tema y se añaden nuevas operaciones para cumplir con las restricciones de cada problema.

Una de las etapas más importantes de un algoritmo de búsqueda local es la exploración de la vecindad de la solución actual. Una vecindad del VRP está determinada por el criterio de vecindad y una solución inicial, llamada solución actual. Sin embargo, una vecindad del VRP posee otras características como: el número de soluciones en la misma, el número de soluciones que son mejores que la solución actual, el óptimo de la vecindad, la cantidad de óptimos que existen, entre otras.

Un previo conocimiento de la cardinalidad de la vecindad que se desea explorar permite la selección de una estrategia de exploración que se ajuste en mayor medida a la vecindad en cuestión. Por lo tanto, en la siguiente sección se presenta una estrategia para contar eficientemente el número exacto de soluciones en una vecindad cualquiera del VRP. En este contexto eficiente significa que no es necesario construir todas las soluciones vecinas.

En este capítulo se ha expuesto toda la teoría que sirve de base al trabajo realizado. Primeramente, se presenta el VRP, así como las variantes sobre las que se basa este estudio (MDVRP y HFVRP). Posteriormente, se introduce una solución alternativa a la idea clásica pero intratable de probar todas las soluciones posibles buscando la mejor: las metaheurísticas de búsqueda local. En el capítulo siguiente se presenta el sistema al que tributa esta investigación.

Capítulo 2

Sistema

En el presente capítulo se profundiza sobre el sistema ya implementado al que tributa esta investigación. Entre los temas analizados está la variante metaheurística empleada, presentada en la sección 2.1. Para implementar esta estrategia, se necesita poder generar soluciones nuevas a partir de una solución base, sobre esto se profundiza en las secciones restantes. En la sección 2.2 se propone un algoritmo para el cálculo de la cardinalidad de una vecindad, que genera el árbol de vecindad sobre el que se profundiza en la sección 2.3. A continuación en la sección 2.4 se analiza la implementación de las funciones que permiten dado un número natural cualquiera, obtener la solución correspondiente dentro del árbol. Finalmente, en la sección 2.5 se emplean las ideas anteriores para generar soluciones vecinas del VRP.

2.1. Búsqueda de Vecindad Infinitamente Variable

En el sistema que se desarrolla en este trabajo se emplea la técnica de Búsqueda de Vecindad Infinitamente Variable. Es una metaheurística que se basa en el cambio sistemático de entorno dentro de una búsqueda local, siguiendo los siguientes principios:

- Un mínimo local con una estructura de entorno no lo es necesariamente en otra.
- Un mínimo global es mínimo global con todas las posibles estructuras de entorno.

De forma que si se tiene una solución que no puede ser mejorada con ninguno de los infinitos criterios de vecindad, se puede tener la certeza de que esa solución es un óptimo global.

El cambio sistemático de infinitos criterios de vecindad es posible gracias a la generación infinita de los mismos lograda en [8].

En pos de la eficiencia, se realizan análisis previos de la vecindad, antes de escoger la siguiente estructura de entorno a aplicar. En la siguiente sección se introduce el concepto de cardinalidad de la vecindad, así como una implementación del cálculo de la misma.

2.2. Cardinalidad de las operaciones

La cardinalidad de una vecindad es el número de soluciones vecinas que la componen. En la creación de cada una de estas nuevas soluciones, intervienen las operaciones elementales, y cada una de ellas genera un número diferente de vecinos, en dependencia del cambio que aplican a la solución actual. Por ejemplo, la selección de cliente, implica la creación de tantos vecinos como clientes tenga la ruta en la que se aplica, pues cada opción disponible, representa una nueva solución. Entonces, se puede crear una algortimo que de forma recursiva, calcule la cardinalidad de cada operación, donde el resultado final sea la cardinalidad de la vecindad.

Para realizar este cálculo las operaciones principales se clasifican en tres grupos: principales, modificadoras y pasivas [11].

Las operaciones **principales** son aquellas que definen el valor de ciertas propiedades comunes para varias operaciones del criterio. Un ejemplo de este tipo de operaciones es la operaci/'on de selección de ruta, que define la selección de un cliente posteriormente.

Las operaciones **modificadoras**, por su parte, son las que modifican la solución actual. Ejemplo de estas operaciones son la selección, inserción o intercambio de clientes dentro de las rutas, pues en cada uno de estos casos se modifica el tamaño de la ruta. En el caso de la selección de cliente, este se extrae de la ruta, con lo que se disminuye en uno la cantidad de clientes de la misma. Las otras dos operaciones (inserción e intercambio) aumentan en uno la cardinalidad de la lista de clientes de las rutas que intervienen. La inserción porque se agrega un nuevo cliente y el intercambio porque se traduce en realizar dos inserciones.

Las operaciones **pasivas**, son las que aportan a la cardinalidad de la

vecindad de la solución actual, o sea, al aplicarlas el valor de la cardinalidad se mantiene igual, como intercambiar depósitos en problemas de múltiples depósitos.

Si se emplea el critero del ejemplo 5, se puede observar que en él se tienen como:

- **Operaciones Principales:** seleccionar ruta r_1, seleccionar ruta r_2.
- **Operaciones Modificadoras:** seleccionar cliente c_1, seleccionar cliente c_2, intercambiar c_1 y c_2.
- **Operaciones Pasivas:** no existen en este criterio

De las clasificaciones anteriores, resulta necesario destacar que las mismas no son excluyentes. Esto significa que una operación puede pertenecer a varias de estas clasificaciones.

La lista con las operaciones elementales de un criterio dado, con los valores que puede tomar cada operación, se denomina instanciación de la operación y representan una solución vecina: la que se obtiene al aplicarle a la solución actual las operaciones indicadas.

Por ejemplo, en el caso mostrado en la figura 1.2 se le pueden asignar los siguientes valores a las operaciones.

- $r_1 = 1$: La ruta de la cual se extraerá el cliente c_1 es la primera ruta de la solución dada.
- $c_1 = 1$: De los posibles clientes a escoger dentro de la ruta 1, se selecciona el primero
- $r_2 = 2$: La ruta r_2 en la cual se seleccionará el cliente c_2 es la segunda ruta de la solución
- $c_2 = 1$: Se seleccionará el cliente que ocupa la primera posición de la ruta r_2
- $c_1.insertion_pos = c_2.selection_pos$ y $c_2.insertion_pos = c_1.selection_pos$ para la operación de intercambio de clientes.

Si se aplica entonces el criterio de la figura 1.2 a la solución de la figura 1.1 se obtiene la solución que se muestra en la figura 2.1.

Si se desea calcular la cardinalidad de las operaciones que intervienen en este criterio, o de cualquier otra, se deben seguir los siguientes pasos:

$$\begin{array}{ll} \text{S} = & ((4\ 2\ 3) \\ & (1\ 5) \\ & (6)) \end{array}$$

Figura 2.1: Ejemplo de solución CVRP después de aplicado el criterio de la figura 1.2

$$S_c = (1,\ 2,\ 3)$$

Figura 2.2: Ejemplo de solución CVRP comprimido basado en la figura 1.1

1. Definir el conjunto de posibles instanciaciones para la operación.
2. Actualizar el estado de la solución, o sea, modificar la solución actual siguiendo lo dictado por las operaciones del criterio.
3. Combinar cada una de las instanciaciones obtenidas, con las restantes generadas por cada operación.

Cabe resaltar también, que para disminuir la complejidad temporal, se elimina toda la información no relevante para el número de soluciones vecinas, basado en la figura 1.1, la solución de conteo (S_c) que se analiza es la que se observa en la figura 2.2.

La solución de conteo no necesita tener información sobre qué clientes están en cada ruta, sino cuántos son. Es por ello que la solución de conteo de la figura 2.1 es también la que se muestra en la figura 2.2.

Entonces, para calcular la cardinalidad de la operación selección de ruta, se define la función de la figura **??**.

En este caso, la cantidad de soluciones que define la operación de selección de ruta es el resultado de la suma de las posibles soluciones que se generan en dependencia de la ruta escogida.

Si se analiza la selección de un cliente, esta es una operación modificadora, esto quiere decir que generan cambios directamente en la cardinalidad de la operación, como se vio en el capítulo anterior.

En este caso, se necesita saber cuántas selecciones es posible realizar, o sea, cuántos clientes hay en la ruta, esta será la contribución de la operación de selección de cliente a la cardinalidad de la vecindad. Una vez seleccionado un cliente cualquiera (decrementado el valor de la ruta en la solución de conteo), se calcula la cantidad de soluciones que generan sus hijos. El producto

```
     COUNT_NEIGHBORS(op, sol, other_ops):
1    count = 0
2    for op.route.id from 1 to sol.routes.len:
3         next_op = other_ops[0]
4         rest_op = other_ops[1:]
5         count += COUNT_NEIGHBORS(next_op, sol, rest_ops)
6    return count
```

Figura 2.3: Función de cálculo de cardinalidad de la operación **Selección de ruta**

```
\begin{verbatim}
     COUNT_NEIGHBORS(op, sol, other_ops):
1    posibilities = op.route.len
2    sol.routes[op.route.id] -= 1
3    next_op = other_ops[0]
4    rest_op = other_ops[1:]
5    count = COUNT_NEIGHBORS(next_op, sol, rest_ops)
6    sol.routes[op.route.id] += 1
7    return posibilities * count
```

Figura 2.4: Función de cálculo de cardinalidad de la operación **Selección de cliente**

de ambos resultados es la cantidad de vecinos generados a partir de ella. Al final del cálculo se retorna la solución actual a su estado inicial.

En [11] se presentan las funciones para calcular la cardinalidad de cada una de las operaciones aplicables en un CVRP.

Aprovechando la definición recursiva de estas funciones es posible definir un árbol de vecindad que almacene y refleje la ejecución de dicho algoritmo. Este se presentará en la siguiente sección.

2.3. Árbol de Vecindad

El árbol de vecindad se define a partir de una estructura de entorno y una solución inicial. Cada rama desde la raíz hasta la hoja representa el conjunto

$$\begin{aligned} S = \quad & ((1\ 2) \\ & (3\ 4\ 5)) \end{aligned}$$

Figura 2.5: Ejemplo de solución CVRP

1. Seleccionar ruta r_1
2. Seleccionar cliente c_1
3. Insertar cliente c_1 en la ruta r_1

Figura 2.6: Ejemplo de estructura de entorno

de decisiones o instanciaciones de operaciones que fue necesario realizar para llegar a la solución vecina representada en la hoja.

Cada nodo almacena una referencia a su hijo (operación que le sigue en la estructura de entorno) y el número de soluciones que hay en el subárbol del cual él es raíz. Además, cada uno de estos nodos guarda información relevante sobre su propia naturaleza, por ejemplo, el nodo de selección de cliente, guarda la ruta donde se aplicará la selección y el número de clientes que hay en ella.

Si se define la solución de la figura 2.5 y el criterio de la figura 2.6, para cada una de estas operaciones, se crea un nodo correspondiente dentro del árbol de vecindad.

- r-node: Operación seleccionar ruta
- a-node: Operación seleccionar cliente
- b-node: Operación insertar cliente

Si se aplica la estructura de entorno de la figura 2.6 a la solución de la figura 2.5 se obtiene árbol de la figura 2.7.

Entre paréntesis se muestra el número de vecinos que tiene el subárbol del que cada nodo es raíz.

Los nodos **end** se emplean para mostrar que se llegó al final del criterio, o sea, se tomaron todas las decisiones necesarias para instanciar esa rama.

Se aprecia que la cantidad de nodos del árbol crece exponencialmente con respecto al número de rutas y de clientes en ellas, lo que hace que su

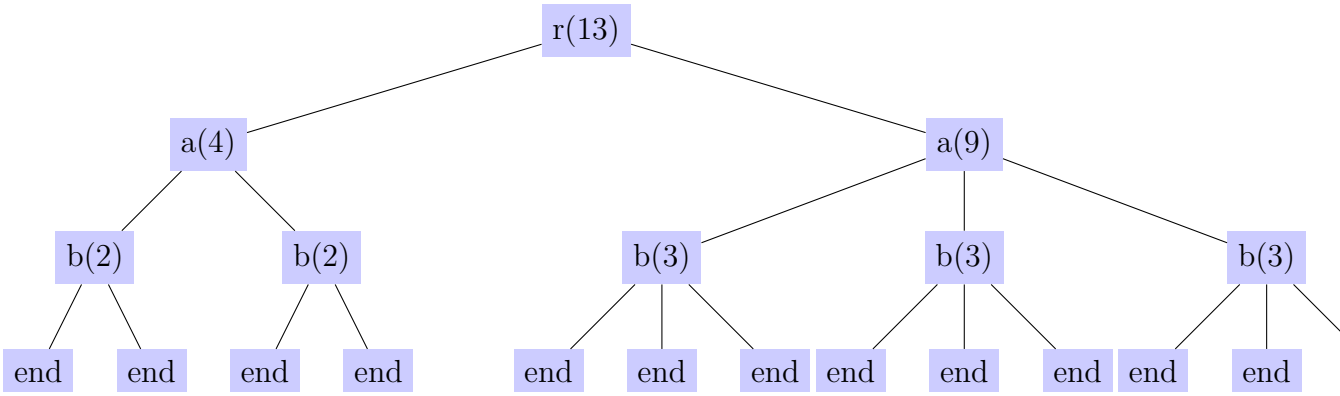

Figura 2.7: Árbol de vecindad expandido para el entorno de la figura 2.6 aplicado sobre la solución de la figura.

2.5

construcción sea similar a analizar la vecindad entera. Por ese motivo se decide simplificar el árbol de vecindad y definirlo como se muestra en la figura 2.8.

Este árbol de vecindad comprimido, permite guardar la misma información en una estructura mucho más compacta, que minimiza la complejidad espacial de su almacenamiento y la complejidad temporal de su análisis. Una de las ventajas de ese árbol es que permite calcular eficientemente una correspondencia uno a uno entre las soluciones vecinas y los números naturales entre 1 y N, donde N es la cardinalidad de la vecindad. A continuación, se presenta cómo calcular esta correspondencia.

2.4. Indexación

Como se ha visto, es posible definir una solución a partir de otra, usando un criterio de vecindad instanciado. Esta idea permite ordenar las soluciones dentro de un entorno de acuerdo a los índices asignados a cada operación elemental.

Para lograr instanciar un criterio es necesario asignar un valor válido a las propiedades de cada operación elemental que lo componen.

En este sentido, la primera solución de la vecindad se define como la solución generada a partir de la instanciación del criterio donde se le asigna el menor valor posible a todas propiedades de las operaciones elementales.

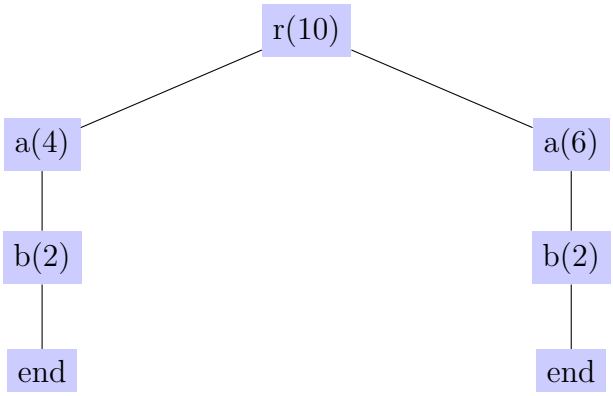

Figura 2.8: Árbol de vecindad contraído para el entorno de la figura 2.6 aplicado sobre la solución de la figura 2.5

$$\begin{array}{ll} \text{S} = & ((2\ 1\ 3) \\ & (4\ 5) \\ & (6)) \end{array}$$

Figura 2.9: Ejemplo de solución CVRP

En el ejemplo de la figura 1.2, la primera instancia de ese entorno sería:

$$\{R_1 = 1,\ c_1 = 1,\ R_2 = 1,\ c_2 = 1\}.$$

Si se aplica a la solución de la figura 1.1, la solución resultante sería la obtenida a partir de la selección del cliente que ocupa la primera posición de la ruta 1 y el intercambio de este con el cliente que ocupa la primera posición de la ruta 1. En este caso se obtiene una solución idéntica a la solución inicial.

Para hallar la segunda solución se incrementa en uno la última operación de la primera instanciación. Para el ejemplo anterior sería

$$\{R_1 = 1,\ c_1 = 1,\ R_2 = 1,\ c_2 = 2\}$$

y la solución generada sería la resultante de intercambiar el primer cliente de la primera ruta con el segundo cliente de la primera ruta, ofreciendo como resultado la solución de la figura 2.9

Cuando la última operación llega al máximo valor posible, se inicializa a su valor mínimo y se incrementa la penúltima operación, y así continúa el procedimiento hasta llegar al máximo de operaciones posibles.

Si se desea, dado un número n, obtener la solución correspondiente, se recorre el árbol de vecindad aplicando el algoritmo de búsqueda n-aria.

Por ejemplo, si se busca la solución número diez en el árbol del ejemplo 3, se comienza aplicando el algoritmo de búsqueda n-aria desde la raíz.

Se conoce que la raíz tiene trece hijos, por tanto, es posible realizar la indexación (si el número n, índice escogido, es mayor que la cantidad de soluciones no tiene sentido realizar el análisis). El hijo de la izquierda (correspondiente a escoger la primera ruta) permite la generación de cuatro vecinos, como diez es mayor que cuatro, entonces esta no es la rama correcta, se mueve el análisis a la siguiente rama, pero descontando del índice general la cantidad de soluciones que se han dejado atrás. O sea, ahora se busca la solución seis dentro del árbol que tiene al nodo **a(9)** como raíz.

En esta rama se tiene que cada uno de sus hijos genera tres nuevos vecinos, por tanto, tampoco se puede escoger el primero, dejando como resto tres. El segundo hijo genera tres nuevas soluciones, por tanto, este es el camino que se debe escoger. La tercera solución dentro de esta rama es la solución número diez de la vecindad. Entonces, la instanciación del criterio que permite encontrar esta solución es: $\{r = 2,\ a = 2,\ b = 3\}$, o sea, seleccionar dentro de la ruta 2, el cliente que se encuentra en la posición 2 e insertarlo en la posición 3.

En el análisis de las vecindades, resulta conveniente particionar las vecindades en regiones con características similares e intentar identificar algunas de las propiedades que pueden presentarse en estos grupos.

En [11] se propone particionar las vecindades en clases de equivalencia compuestas por el conjunto de soluciones de la vecindad que poseen valores iguales para las operaciones principales.

Partiendo de esta idea, se puede definir una nueva forma de indexación similar a la anterior pero donde las operaciones principales serán las últimas en ser incrementadas.

Esto quiere decir que, al igual que en criterio anterior el primer vecino será el obtenido a partir de $\{R_1 = 1,\ c_1 = 1,\ R_2 = 1,\ c_2 = 1\}$, y luego se incrementará el valor de c_2 hasta llegar a su máximo valor. Posteriormente se aumentará la primera operación a la izquierda no principal, o sea, el valor de c_1. Al llegar al máximo de todas las operaciones no principales, entonces se comienza a incrementar las principales siguiendo el mismo orden.

Para la creación del árbol, y el análisis del mismo, se ordenan todas las soluciones principales en las primeras ramificaciones de árbol, y debajo se ubican las operaciones modificadoras y pasivas en el orden en que aparezcan

```
     EXHAUTIVE_EXPLORATION(neigh):
1    card_neigh = COUNT_NEIGHBORS(neigh)
2    solutions = []
3    for i from 1 to card_neigh:
4         cur_sol = NTH_NEIGHBOR (neigh, i)
5         solutions = solutions + {cur_sol}
6    return solutions
```

Figura 2.10: Exploración Exhaustiva

en el criterio.

Las funciones de indexación presentadas en esta sección constituyen un elemento fundamental en los generadores definidos en la herramienta analizada, sobre los que se profundiza en la siguiente sección.

2.5. Generadores

En el proceso de exploración de una vecindad se parte de una solución inicial y una estructura de entorno. Este proceso consiste en iterar sobre las soluciones vecinas que se generan de acuerdo a cierto criterio (llamado estrategia de exploración), analizar qué tan buena es cada una y seleccionar una de ellas a través de algún otro criterio (llamado estrategia de selección).

Las técnicas clásicas son la exploración exhaustiva y la exploración aleatoria.

En el primer caso, se analizan todas las soluciones de una vecindad en un orden determinado. En el sistema que utiliza este trabajo, dicho orden está definido por la función de indexación que se utiliza para la vecindad en cuestión. En la figura 2.5 se presenta la función que representa esta estrategia.

Por otro lado, en la exploración aleatoria, se analiza un subconjunto aleatorio de todas las posibles soluciones de una vecindad. El número de elementos de dicho subconjunto está definido por el usuario y el orden de los mismos está determinado por un ordenamiento aleatorio de los elementos de la vecindad. Se define entonces la función de la figura 2.5.

En el sistema se introducen además dos nuevas estrategias de exploración: combinatoria y secuencial.

```
    UNIFORM_EXPLORATION(neigh, cant):
 1  card_neigh = COUNT_NEIGHBORS(neigh)
 2  permutation = PERMUTACION_ALEATORIA(1, card_neigh)
 3  solutions = [ ]
 4  for i in permutation:
 5      if cant == 0:
 6          return solutions
 7      cur_sol = NTH_NEIGHBOR (neigh, i)
 8      solutions = solutions + {cur_sol}
 9      cant -= 1
10   return solutions
```

Figura 2.11: Exploración Aleatoria

El primer caso, consiste en explorar las soluciones de la misma, de tal forma que se cumpla que entre dos soluciones que pertenezcan a una misma región de la vecindad exista exactamente una solución por cada una de las regiones restantes que no se han analizado completamente.

En cuanto al orden a seguir, debe cumplirse que la región de la próxima solución generada, será aquella que le sigue en dicho orden a la región de la solución anterior y aún cuenta con soluciones por ser analizadas. En el caso de alcanzar la última región en el orden, es necesario comenzar nuevamente por la primera región hasta que todas las regiones de la vecindad hayan sido completamente analizadas. Esta exploración de la vecindad se puede representar como muestra la figura 2.5.

La función IS_EXHAUSTED se emplea en la figura 2.5 para verificar si se ha analizado toda la vecindad.

Para el caso de exploración secuencial todas las soluciones de una región deben ser analizadas antes de continuar el proceso de exploración de la siguiente región siguiendo un orden determinado. Este orden no tiene por qué contener a todas las regiones de la vecindad, por lo que se define como un subconjunto ordenado de las regiones de la misma. El siguiente fragmento de pseudo-código define dicha exploración.

En este capítulo se presenta el sistema al que tributa esta investigación. En la sección 2.1 se profundizó sobre la metaheurística de búsqueda local empleada en el sistema que se construye, la Búsqueda de Vecindad Infinitamente Variable.

```
    COMBINATORIAL_EXPLORATION(neigh):
 1  generators = [ ]
 2  for reg in neigh.regions:
 3      cur_gen = UNIFORM_EXPLORATION(reg)
 4      generators = generators + {cur_gen}
 5  solutions = [ ]
 6  while True:
 7      if IS_EXHAUSTED(neigh):
 8          break
 9  for gen in generators:
10      cur_sol = gen()
11      if cur_sol != "nil":
12           solutions = solutions + {cur_sol}
13  return solutions
```

Figura 2.12: Exploración Combinatoria

```
    SECUENCIAL_EXPLORATION(neigh):
 1  generators = [ ]
 2  for reg in reg_lst:
 3      cur_gen = UNIFORM_EXPLORATION(reg)
 4      generators = generators + {cur_gen}
 5  solutions = [ ]
 6  for gen in generators:
 7      while True:
 8          cur_sol = gen()
 9          if cur_sol != "nil":
10              break
11          solutions = solutions + {cur_sol}
12  return solutions
```

Figura 2.13: Exploración Secuencial

Posteriormente se introdujeron conceptos fundamentales para poder usar esta estrategia: criterios de vecindad y operaciones elementales así como la clasificación de estas en dependencia de si generan o no cambios en la cardinalidad de una vecindad.

Finalmente se discutieron algunas estrategias para la generación de soluciones a partir de estos criterios. Primeramente se mostró cómo calcular la cardinalidad de cada operación. Por último, se presentó una forma de indexar las soluciones de la vecindad y posteriormente, se introducen los generadores empleados en el sistema.

En los capítulos siguientes describe cómo extender el sistema para realizar todos estos procesos en problemas que incluyen múltiples depósitos y múltiples vehículos.

Capítulo 3

VRP con depósito múltiple

La variante de VRP con múltiples depósitos (MDVRP, por sus siglas en inglés) tiene gran similitud al problema clásico y solo se diferencia de este por la existencia de varios depósitos que pueden ser empleados a la hora de diseñar las rutas. Como se mencionó en secciones anteriores, si estos depósitos están aislados entre sí y se pueden seleccionar clientes cercanos al área geográfica de cada uno de ellos, la solución sería resolver varios VRPs clásicos, uno para cada área delimitada. En el caso de que los depósitos y clientes no puedan ser aislados el problema se torna más interesante, y será esta variante la que se analizará en el resto de este capítulo.

Este capítulo tiene una estructura similar al anterior. En la sección 3.1 se presenta la estrucruta de la solución de MDVRP. Posteriormente en la sección 3.2 se definen las operaciones elementales que deben ser agregadas al sistema para resolver este tipo de problemas. Además en la sección 3.3 se brinda la implementación de las funciones de cardinalidad para dichas operaciones, generando nuevos nodos en el árbol de vecindad, los cuales se analizan en 3.4. En la sección 3.5 se presenta el diseño de los algoritmos de indexación dentro del árbol.

3.1. Solución

Una solución del MDVRP requiere de la distribución del conjunto de clientes en una o varias rutas, y de la asignación de un depósito de la lista de depósitos disponibles, a cada ruta. Ese depósito servirá de origen y destino final de cada recorrido.

S = (((1: 1 2 3)
(2: 4 5)
(1: 6))
(1 2 3))

Figura 3.1: Ejemplo de solución MDVRP

- Intercambiar depósitos de dos rutas
- Intercambiar el depósito de una ruta por otro de la lista de depósitos disponibles.

Figura 3.2: Criterios de vecindad relacionados con MDVRP

Una ruta está determinada por un depósito d_i y un conjunto de clientes $\{c_{i1}, c_{i2}, \ldots, c_{iN_i}\}$, donde N_i es la cantidad de clientes en la ruta i. Se define el ejemplo de la figura 3.1. Aquí se muestra una instancia del MDVRP, donde las rutas se representan con el formato (d: c_1, c_2, ... c_n) y el conjunto final guarda los depósitos disponibles.

En la siguiente sección se presentan las nuevas operaciones elementales que se pudieran incluir en criterios de vecindad para modificar soluciones de un MDVRP.

3.2. Operaciones

EL MDVRP tiene un elemento que no está presente en los VRP con depósito único y es la existencia de una lista de depósitos disponibles para la creación de las rutas. Esta nueva característica permite crear estructuras de entorno que modifiquen el depósito asignado a cada ruta generando cambios en la solución inicial, como las que se muestran en la figura 3.2. Las opraciones elementales que permiten definir estas estructuras de entorno se muestran en la figura 3.3.

De acuerdo con el criterio de clasificación presentado en la sección 2.2, se puede decir que las operaciones elementales primera y tercera son pasivas, pues no representan cambios en la cardinalidad de la vecindad. Sin embargo, la segunda operación es modificadora, ya que tiene una lista de posibilidades para seleccionar entre los vehículos disponibles.

1. Seleccioar depósito de una ruta
2. Seleccionar depósito de la lista de depósitos
3. Intercambiar depósitos

Figura 3.3: Operaciones elementales

```
COUNT_NEIGHBORS ( op , sol , other_ops )
1    next_op = other_ops [0]
2    rest_ops = other_ops [1:]
3    count = COUNT_NEIGHBORS ( next_op , sol , rest_ops )
4    return count
```

Figura 3.4: Función de cálculo de la cardinalidad para la operación de selección de depósito dentro de una ruta

Si se desea que el sistema que está implementado en la facultad sea capaz de resolver variantes de VRP que incluyan múltiples depósitos se hace necesario definir e implementar cada una de estas operaciones.

Para ello se debe agregar una serie de clases y métodos que permitan la integración de la nueva operación con todo el trabajo previo. Como aporte teórico de esta investigación se tiene la forma de calcular la cardinalidad de cada una de estas operaciones que se presentan en la sección siguiente.

3.3. Cálculo de la cardinalidad

El cálculo de la cardinalidad de una operación se hace vital en la herramienta propuesta, pues permite tener un conocimiento previo del tamaño de la vecindad para decidir cómo se continuará el análisis.

Al agregar una nueva operación los generadores necesitan conocer cómo calcular el número de soluciones vecinas que se puede obtener aplicando dicha operación. La forma en la que fue definido este proceso dentro del sistema, hace que solo sea necesario agregar funciones que calculen la cardinalidad de la operación agregada. En la figura 3.3 se define la función de conteo para la operación de selección de depósito dentro de una ruta.

Esta operación no modifica el número de soluciones vecinas, pues solo

```
COUNT_NEIGHBORS ( op , sol , other_ops )
1    posibilities = sol . non-selected-depots
2    next_op = other_ops [0]
3    rest_ops = other_ops [1:]
4    count = COUNT_NEIGHBORS ( next_op , sol , rest_ops )
5    return possibilities * count
```

Figura 3.5: Función de cálculo de la cardinalidad para la operación de selección de depósito de la lista de depósitos disponibles.

tiene una posibilidad de efectuarse, así que el número de vecinos solo depende de las siguientes operaciones en la lista del criterio de vecindad.

La operación de intercambio de depósitos, al igual que la selección de depósitos dentro de una ruta, tiene solo una forma de aplicarse, que estará definida por los dos depósitos previamente seleccionados. Es por ello que su función de cardinalidad se calcula también como muestra la figura 3.3.

En el caso de la selección de un depósito de la lista, esta operación se puede instanciar de tantas formas como depósitos estén sin usar en ese momento del problema. Esto se refleja en el código de cálculo de vecinos:

De esta forma se puede determinar la cardinalidad de cualquier vecindad que use estas operaciones. Sin embargo, para explorar la vecindad es necesario determinar cómo obtener cada vecino y esto se hace a partir del árbol de vecindad. Para generar vecinos de un criterio que incluyan operaciones con depósitos, es necesario implementar los nodos que representan estas operaciones en el árbol. Ese es el objetivo de la siguiente sección.

3.4. Árbol de vecindad

Para agregar las operaciones con depósitos al árbol de vecindad es necesario definir nuevos nodos:

- **d-node**: Este nodo representa la operación de seleccionar un depósito dentro de una ruta. Almacena una referencia a su hijo en el árbol, así como la ruta en la que se realizará la selección.

- **x-node**: Este nodo representa la operación de seleccionar un depósito de la lista de depósitos disponibles. Almacena una referencia a su hijo.

- **s-node**: Este nodo representa la operación de intercambiar dos depósitos, ya sea depósitos de dos rutas previamente seleccionadas o de una ruta y la lista de depósitos disponibles. Guarda una referencia al nodo hijo, así como el la identificación de las operaciones de selección de depósito previas.

Estos nodos permiten que las nuevas operaciones intervengan en análisis más complejos como la generación de soluciones vecinas a través del proceso de indexación. Este es el tema tratado en la siguiente sección.

3.5. Indexación

La indexación es una parte fundamental para la exploración de la vecindad usando generadores. Al algregar una nueva operación es necesario implementar una función compatible con el mecanismo ya existente y que permita indexar en cualquier árbol de vecindad. O sea, esto significa que dado un número entre uno y la cantidad de soluciones en la vecindad, determinar cuál es la solución que le corresponde a ese número.

Para cada nodo se implementa una versión de NTH_NEIGHBORS, ya que las características del nodo influyen en el cálculo que debe ser realizado.

Para entender cada una de las implementaciones que se mostrarán es necesario definir:

- **child:** se emplea para hacer referencia al hijo del nodo actual.
- **total:** se emplea para calcular el número de soluciones generadas a partir de un nodo del árbol de vecindad.
- **'d:** hace referencia a la operación de selección de depósito.
- **'s:** hace referencia a la operación de selección de ruta.
- **'x:** hace referencia a la operación de selección de depósito dentro de la lista de depósitos disponibles.
- **route-number:** campo que guarda el número de una ruta previamente seleccionada.
- **node.ID:** identificador asignado a cada nodo.

```
NTH_NEIGHBORS( d-node , index )
1  if index > total(child)
2   error: "El índice no es válido"
3  else
4   new-op = list( 'd , route-number , node.ID)
5   return (CONCAT ( new-op , NTH_NEIGHBORS(child, index))))
```

Figura 3.6: Función de indexación para la operación seleccionar depósito en una ruta.

En la figura 3.5 se define la función NTH_NEIGHBORS para el nodo seleccionar un depósito dentro de una ruta. En un primer paso, analiza si el índice es válido. De serlo, se crea la estructura de la nueva operación y se devuelve la concatenación de esta con el resultado obtenido de evaluar NTH_NEIGHBORS en el hijo. Esta operación no modifica la cardinalidad de la vecindad, por lo que no es necesario realizar otros cálculos.

En el caso de la operación seleccionar un depósito de la lista de depósitos disponibles, el análisis se hace más complejo pues esta selección tiene más de un valor posible. En estos casos se debe analizar qué implica la selección de cada uno de los depósitos disponibles para la cardinalidad de la vecindad.

Si se aplica el criterio de la figura 3.5, sobre la solución de la figura 3.1 se obtiene el árbol de vecindad que se muestra en la figura 3.8. Aquí se puede apreciar que la operación tiene tantos hijos como depósitos disponibles en el problema, y se debe determinar en cuál de esas ramas se encuentra la solución deseada. Cada una de estas ramas tiene la información de cuántas soluciones se pueden obtener a partir de ella, y esa información se emplea para determinar la que contiene al vecino buscado.

Para este ejemplo, si se desea obtener la solución número ocho, se comienza analizando desde la raíz. En este nodo se verifica la cantidad de soluciones que contiene cada uno de sus hijos, y se concluye que la solución buscada debe estar en el tercer hijo de la raíz. Así se va bajando por el árbol hasta llegar a la solución buscada.

En el caso del árbol sobre el que estamos trabajando(árbol de vecindad contraído), la operación tiene un único hijo pero, basados en la idea de que todos los hijos genereran la misma cantidad de soluciones se puede decir que NTH_NEIGHBORS para el nodo x-node se define como muestra la figura 3.5. La instancia de la operación que permite crear la solución buscada será

1. Seleccionar ruta r_1.
2. Seleccionar depósito d_1 en la ruta r_1.
3. Seleccionar depósito d_2 de la lista de depósitos disponibles
4. Intercambiar d_1 y d_2

Figura 3.7: Criterio de vecindad que implica el intercambio de un depósito perteneciente a una ruta con otro que pertenece a la lista de depósitos disponibles.

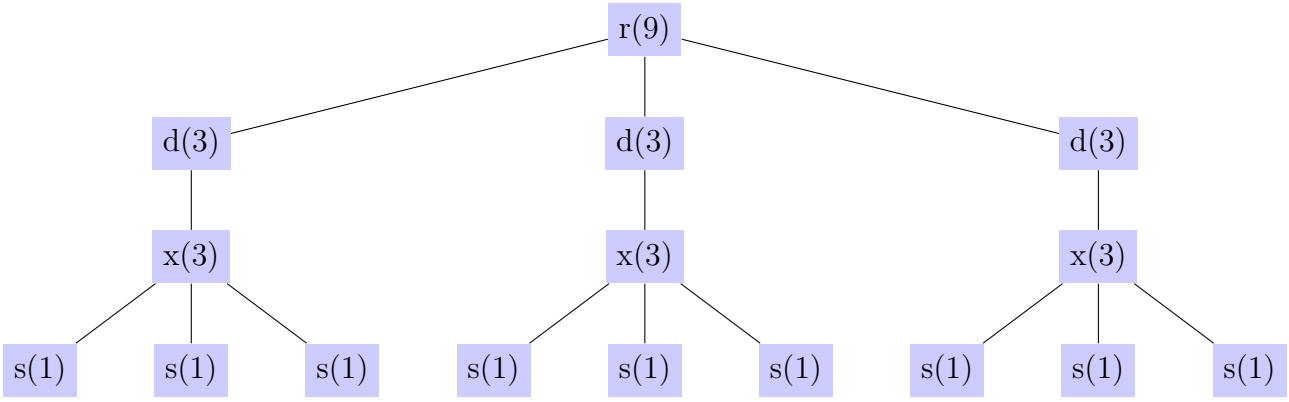

Figura 3.8: Árbol de vecindad expandido para el entorno de la figura 3.5 aplicado sobre la solución de la figura 3.1.

la resultante de dividir el índice que se recibe de entrada y el número de valores que puede tomar la operación, redondeado por defecto. Al final de la función se concatena la operación creada con las creadas por los hijos. El llamado a NTH_NEIGHBORS del hijo se realiza con el nuevo índice que se obtiene de restar al índice recibido las operaciones que fueron descartadas.

En el caso de la operación de intercambio, la función NTH_NEIGHBORS se hace muy similar a la del d-node, pues en todas las instancias de esta operación solo tiene una alternativa:

```
NTH_NEIGHBORS( x-node , index )
1 pos = int(index / total(child)) + 1
2 if index == total(child) * (pos - 1)
3 pos = pos -1
4 if pos > x-node.possibilities:
5 error: "El índice no es válido"
6 else
7 new-op = list( 'x , node.ID)
8 new-index = index - (( pos - 1 ) * total(child))
8 return (CONCAT ( new-op , NTH_NEIGHBORS( new-index, child)))
```

Figura 3.9: Función de indexación para la operación seleccionar depósito en una ruta

```
NTH_NEIGHBORS( s-node , index )
1  if index > total(child)
2   error: "El índice no es válido"
3  else
4   new-op = list( 's , select-op1.ID , select-op2.ID)
5   return (CONCAT ( new-op , NTH_NEIGHBORS(child, index))))
```

Figura 3.10: Función de indexación para la operación de intercambio de depósitos.

3.6. Ejemplo

Las implementaciones propuestas permiten que el sistema al que tributa esta investigación sea capaz de resolver instancias de VRP que incluyan características de CVRP y MDVRP.

En la figura 3.11 se define un criterio de vecindad que modifica la posición de un cliente dentro de una ruta y, además el depósito asignado a una ruta.

Si se escoge, como solución actual la presentada en la figura 3.1, entonces se puede calcular la cardinalidad del entorno resultante.

El análisis comienza con el primer elemento del criterio: Seleccionar ruta r_1. Esta operación es principal y en la solución actual se tienen 3 posibles rutas, por lo que el árbol de la vecindad tiene como raíz un **r-node**, que a su vez tendrá tres hijos. El valor de la cardinalidad de esta operación es la suma de los valores calculados por sus hijos, de acuerdo a lo planteado en la figura 2.3.

La siguiente operación elemental en el criterio es: Seleccionar cliente c_1 en la ruta r_1. Por lo que cada uno de los hijos de las raíz será un **a-node**. Esta operación es modificadora, esto significa que tendrá un único hijo. El valor de la cardinalidad de esta operación depende de la cantidad de valores que puede tomar la propiedad que la define, como plantea la función presentada en el código de la figura 2.4.

Su hijo será un **b-node**, que será quien represente la tercera operación elemental del criterio analizado. La cardinalidad de esta operación depende del número de posiciones que se puedan escoger dentro de la ruta [11], y se calcula de manera similar a la operación anterior.

La siguiente operación en la lista es un **r-node** que simboliza la segunda elección de ruta. El análisis se hace análogo a la anterior selección de ruta.

Posteriormente se tiene una selección de depósito de una ruta. Para calcular la cardinalidad de esta operación se usa la función definida en la figura 3.3. Esta operación también tiene un único hijo, que en este caso será un **x-node**.

La operación de selección de depósito dentro de la lista de depósitos disponibles tiene tres posibilidades, una por cada depósito disponible. En este caso, la cardinalidad se calcula como se indica en la función planteada en la figura 3.3.

La última operación de este árbol es intercambio de depósito. Esta es una operación pasiva, por lo que no genera nuevas soluciones.

De esta manera se desciende creando el árbol que se observa en la figura

1. Seleccionar ruta r_1.
2. Seleccionar cliente c_1 en la ruta r_1.
3. Insertar cliente c_1 en la ruta r_1.
4. Seleccionar ruta r_2.
5. Seleccionar depósito d_1 en la ruta r_2.
6. Seleccionar depósito d_2 en la lista de depósitos disponibles.
7. Intercambiar d_1 y d_2.

Figura 3.11: Criterio de vecindad

3.12. En este, se observa que cada nodo tiene asignado un par (x, y), donde las x representan la cantidad de opciones que brinda una operación y las y, la cantidad de vecinos que se generan a partir del subárbol del cual es raíz ese nodo.

A medida que se desciende se actualizan los valores x y se evalúa la función de conteo en los hijos de cada nodo. El resultado será asignado a los valores y, para terminar el cálculo de la cardinalidad. Al llegar a las hojas estas retornan uno, y en los retornos de la recursividad se reciben los valores calculados de todos los nodos hijos.

Finalmente, en el nodo de la raíz, queda el valor esperado: 126.

A partir de este árbol se puede obtener cualquier solución de la vecindad, aplicando las funciones NTH_NEIGHBOR.

Si se desea obtener la solución número cien de la vecindad se comienza el análisis por el nodo raíz como se explica en [11].

El primer hijo de la raíz genera ochenta y una soluciones, como es menor que el índice buscado, este camino no es el que se debe seguir. El segundo hijo tiene treinta y seis soluciones, esto significa que hasta él se pueden contar ciento diecisiete soluciones, como es mayor que el índice recibido, este es el camino correcto. Se crea la operación selección de ruta y su propiedad se inicializa en dos, pues el segundo camino fue el escogido. La lista que representa la instanciación del criterio toma la forma $\{r_1 = 2\}$. A medida que se desciende en el árbol, se añaden a la lista los nuevos valores.

Ahora se debe analizar qué cliente se debe seleccionar para encontrar la

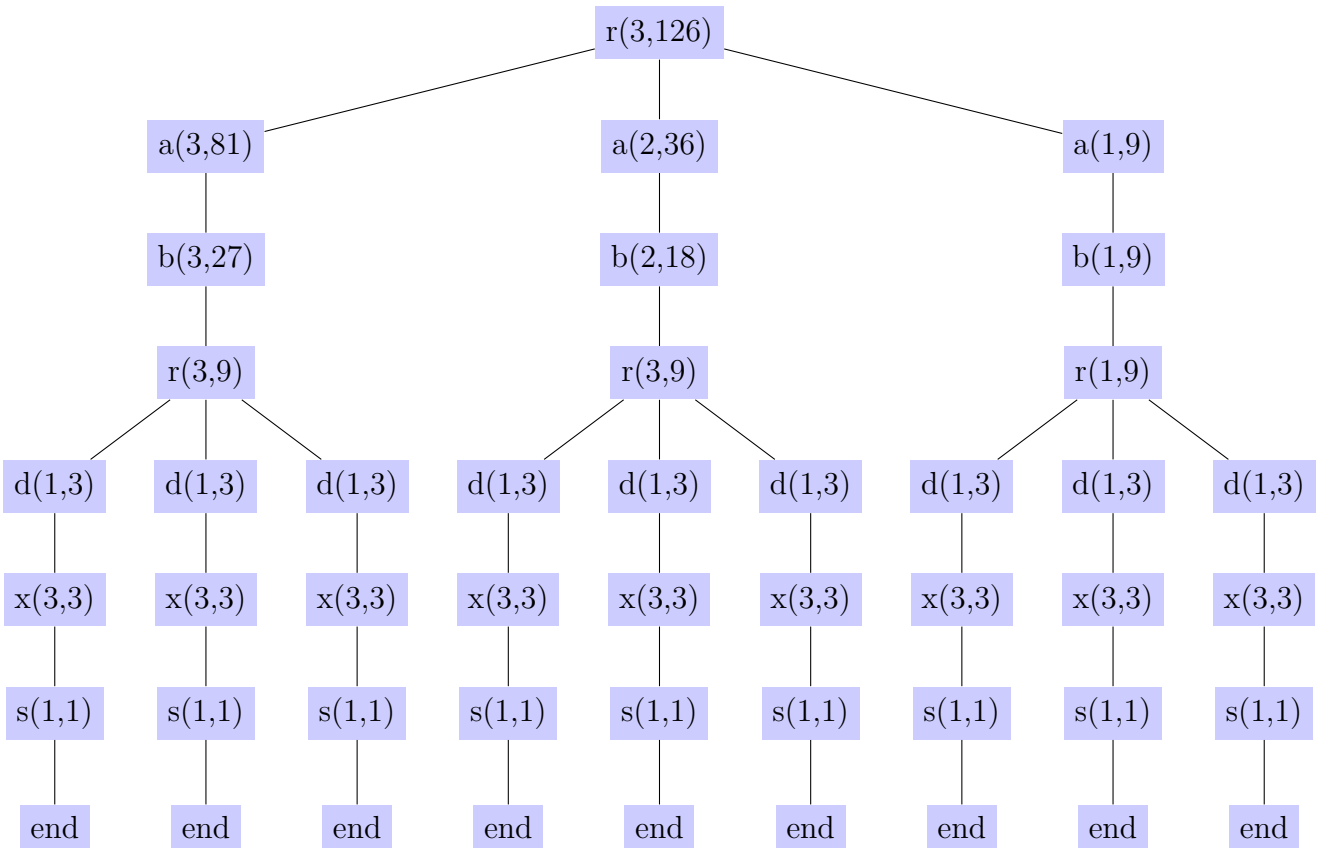

Figura 3.12: Árbol de vecindad expandido para el criterio de vecindad aplicado sobre la solución de la figura 3.1.

solución buscada. Para ellos, se descuenta del índice inicial todas las soluciones descartadas (o sea, el nuevo índice será 100 - 81 = 19). Según [11] la búsqueda se continúa dividiendo el nuevo índice entre el valor obtenido de aplicar la función de conteo al nodo hijo (19 / 18 = 1) y aumentándololo en uno. Entonces el valor de la operación de selección de cliente será dos. Se crea la operación y se sigue buscando. La lista que representa la instanciación del criterio toma la forma $\{r_1 = 2, a_1 = 2\}$.

Ahora se analiza el **b-node**. En este caso, se procede similar al caso anterior. El índice para este nodo es uno (19 - 18 = 1), entonces al dividir entre los valores de los hijos y aumentar en uno ((1 / 9) + 1 = 1), se obtiene que el valor para esta operación será uno. Su valor se agrega a la lista de operaciones instanciadas $\{r_1 = 2, a_1 = 2, b_1 = 1\}$.

El siguiente nodo es la segunda selección de ruta, aquí se procede de forma similar a la primera. El primer hijo tiene una posibilidad y como el índice actual es uno, ese será el camino a escoger. Al agregar a la lista su valor, esta toma la forma $\{r_1 = 2, a_1 = 2, b_1 = 1, r_2 = 1\}$.

El nodo que continúa es selección de depósito dentro de una ruta. Aquí se aplica la función de la figura 3.5, esto significa que como solo hay una alternativa, al ser una operación pasiva, se le asigna el valor uno. La lista de operaciones se actualiza y quedaría $\{r_1 = 2, a_1 = 2, b_1 = 1, r_2 = 1, d_1 = 1\}$

Su hijo, selección de depósito dentro de la lista de depósitos disponibles, tiene como hijo el valor uno. Entonces se aplica el procedimiento definido en 3.5. Se divide el índice actual entre la cantidad de vecinos de sus hijos (1 / 1 = 1) y se cae en un caso particular de este algoritmo, planteado en el paso 2. Entonces, no se incrementa en uno el resultado de la división y se le asigna al valor de la propiedad de la operación analizada en uno. La lista de operaciones recibe este valor y toma la forma $\{r_1 = 2, a_1 = 2, b_1 = 1, r_2 = 1, d_1 = 1, x_1 = 1\}$

Solo resta analizar la operación de intercambio de depósitos como muestra la figura 3.5. Esta operación es pasiva, por lo que se procede similar al **d-node**.

Los nodos **end** se emplean para brindar una caso de parada a este algoritmo recursivo.

Entonces, se obtiene que la instanciación del criterio que devuelve la operación número 100 es:$\{r_1 = 2, a_1 = 2, b_1 = 1, r_2 = 1, d_1 = 1, x_1 = 1, s_1 = 1\}$, obteniéndose la solución de la figura 3.13.

En este capítulo se han expuesto las herramientas necesarias para crear criterios de vecindad afines con el MDVRP y analizar las soluciones resul-

S = (((1: 1 2 3)
((2: 5 4)
((1: 6))
((1 2 3))

Figura 3.13: Ejemplo de solución MDVRP

tantes, en pos de brindar la respuesta óptima a problemas de esta categoría. Las funciones mostradas, representan simplificaciones de las agregadas al sistema al que tributa esta investigación, estas reflejan el funcionamiento de los algoritmos implementados para dar respuesta a los objetivos del trabajo.

En el siguiente capítulo se presenta un análisis similar para los problemas de VRP con Flota Heterogénea.

Capítulo 4

VRP con flota heterogénea

El HFVRP es una variante de VRP donde juega un importante rol la naturaleza de la flota. En estos problemas se cuenta con vehículos diferentes, y la selección de estos para servir una ruta u otra marca una importante diferencia. Estos problemas se clasifican atendiendo al número de vehículos disponibles de cada clasificación en flota infinita(cuando de cada categoría se pueden escoger tantos vehículos como se desee) y flota finita (cuando se tiene una cantidad fija de vehículos de cada categoría).

Este capítulo tiene una estructura muy similar a los anteriores. En la sección 4.1 se presenta la estructura de la solución de HFVRP. Posteriormente en la sección 4.2 se definen las operaciones elementales que deben ser agregadas al sistema para crear estrucuras de entorno que interactúen con las nuevas caractrísticas. Además en la sección 4.3 se brinda la implementación de las funciones de cardinalidad para dichas operaciones, generando nuevos nodos en el árbol de vecindad, los cuales se analizan en 4.4. En la sección 4.5 se presenta el diseño de los algoritmos de indexación dentro del árbol.

4.1. Solución

En estos casos para definir una ruta es necesario especificar el vehículo que será empleado para visitar ese conjunto de clientes, como en el ejemplo de la figura 4.1. En estos casos, la solución está compuesta por el conjunto de rutas y la lista de vehículos disponibles. Cada ruta presenta la siguiente estructura: (v: c_1, c_2, ..., c_n), donde v simboliza el vehículo escogido para visitar a los clientes de la ruta.

S = (((1: 1 2 3)
(2: 4 5)
(3: 6))
(1 2 3))

Figura 4.1: Ejemplo de solución MDVRP

S = (((1,1: 1 2 3)
(2,2: 4 5)
(1,3: 6))
(1 2 3)
(1 2 3))

Figura 4.2: Ejemplo de solución MDVRP

Si se combinan el HFVRP con el MDVRP se obtienen soluciones como la que se encuentra en la figura 4.2. Esta solución guarda el conjunto de las rutas, el conjunto de depósitos disponibles y el conjunto de vehículos disponibles. Cada ruta obedece a la estructura (d,v: c_1, c_2, ..., c_n), donde d y v simbolizan el depósito y el vehículo asignados a la ruta, respectivamente.

4.2. Operaciones

El HFVRP añade una nueva característica al VRP y es la existencia de una lista de vehículos disponibles para la creación de rutas. Esto permite agregar operaciones elementales que posibiliten la definición de estructuras de entorno con el fin de modificar el vehículo asignado a cada ruta. A continuación se presentan estos criterios:

- Intercambiar vehículos de dos rutas
- Intercambiar el vehículo de una ruta por otro de la lista de vehículos disponibles.

Las operaciones elementales que intervienen en estos criterios son:

- Seleccioar vehículo de una ruta.
- Seleccionar vehículo de la lista de depósitos.

```
COUNT_NEIGHBORS ( op , sol , other_ops )
1    next_op = other_ops [0]
2    rest_ops = other_ops [1:]
3    count = COUNT_NEIGHBORS ( next_op , sol , rest_ops )
4    return count
```

Figura 4.3: Función de cálculo de cardinalidad correspondiente a la operación selección de vehículo dentro de una ruta

- Intercambiar vehículos.

Si se analiza el criterio de clasificación presentado en la sección 2.2, se puede decir que las operaciones primera y tercera son pasivas pues no modifican la cardinalidad de la vecindad. La segunda operación es modificadora, pues genera tantos vecinos como vehículos disponibles en el problema.

Si se desea que el sistema al que tributa esta investigación sea capaz de resolver HFVRP es necesario agregar cada una de estas operaciones para poder diseñar las estructuras de entorno antes mencionadas.

Para ello se debe implementar una serie de clases y métodos que permitan la integración de la nueva operación con todo el trabajo previo.

En la siguiente sección, se presentan las funciones de cálculo de la cardinalidad para las operaciones elementales que fueron definidas.

4.3. Cálculo de la cardinalidad

Para cada una de las operaciones definidas en la sección anterior, es necesario implementar la función de cardinalidad.

En la figura 4.3 se presenta la implementación para la selección del vehículo asignado a una ruta. Esta operación no modifica el número de soluciones vecinas, pues puede tomar un solo valor, así que solo depende de las siguientes operaciones en la lista del criterio de vecindad.

De forma análoga se define la función de cálculo de la cardinalidad de la operación de intercambio de vehículos.

En el caso de la selección de un depósito de la lista, esta operación tiene tantas posibilidades como vehículos disponibles en esa instancia del problema. Para el caso de HFVRP, podemos definir esta función como se observa en la figura 4.3.

```
COUNT_NEIGHBORS ( op , sol , other_ops )
1    posibilities = sol . non-selected-depots
2    next_op = other_ops [0]
3    rest_ops = other_ops [1:]
4    count = COUNT_NEIGHBORS ( next_op , sol , rest_ops )
5    return possibilities * count
```

Figura 4.4: Función de cálculo de cardinalidad correspondiente a la operación selección de vehículo de la lista de vehículos disponibles en el HFVRP.

```
COUNT_NEIGHBORS ( op , sol , other_ops )
1    posibilities = sol . vehicles
2    DECREMEMT_VEHICLE( sol,  op)
3    next_op = other_ops [0]
4    rest_ops = other_ops [1:]
5    count = COUNT_NEIGHBORS ( next_op , sol , rest_ops )
6    INCREMEMT_VEHICLE( sol,  op)
7    return possibilities * count
```

Figura 4.5: Función de cálculo de cardinalidad correspondiente a la operación selección de vehículo de la lista de vehículos disponibles en el HFFVRP.

Si la flota se define como finita, entonces escoger un vehículo de la lista de vehículos disponibles, conlleva a eliminarlo de esa lista. Es por ello que la función de conteo de vecinos se define como se muestra en la figura 4.3.

Las funciones $DECREMEMT\_VEHICLE(sol, op)$ e $INCREMEMT\_VEHICLE(sol, op)$ se emplean para modificar la solución existente, al extraer e incluir el vehículo seleccionado de la lista de vehículos disponibles.

El cálculo de la cardinalidad permite tener información previa sobre el número de soluciones dentro de la misma. Esta información se guarda en el árbol de vecindad. En la siguiente sección se explica cómo afectan las operaciones y funciones agregadas al árbol definido en el sistema.

4.4. Árbol de vecindad

Las nuevas operaciones, obligan a implementar nuevos nodos que las representen en el árbol de vecindad. A continuación se presentan los nodos añadidos:

- **v-node:** Este nodo representa la operación de selección de un vehículo dentro de una ruta. Almacena una referencia a su hijo y el valor de la ruta donde se realizará la selección.
- **y-node:** Este nodo representa la operación de selección de un vehículo dentro de la lista de vehículos disponibles para el caso de HFVRP. Almacena una referencia al nodo hijo y el número de vehículos disponibles.
- **z-node:** Este nodo representa la operación de selección de un vehículo dentro de la lista de vehículos disponibles para el caso de HFFVRP. Almacena una referencia al nodo hijo y el número de vehículos disponibles. Difiere del y-node pues en este caso el número de vehículos disponibles es variable.
- **t-node:** Representa la operación de intercambio de vehículos. Este nodo almacena una referencia al nodo hijo y los identificadores de las operaciones de selección.

Estos nodos permiten que las nuevas operaciones intervengan en análisis más complejos como la obtención de soluciones vecinas. Sobre este tema se profundiza en la siguiente sección.

4.5. Indexación

En capítulos anteriores se ha mostrado cómo indexar en el árbol de vecindad, para cada una de las operaciones que se han definido. En la presente sección se presentará la implementación de la función NTH_NEIGHBORS para las operaciones que interviene en ambas variantes de HFVRP.

Las funciones que se presentan a continuación tienen gran similitud a las que intervienen en MDVRP, pues las operaciones relacionadas tienen gran simulitud. Para su comprensión, además de los elementos definidos en 3.5 se necesita conocer:

```
NTH_NEIGHBORS( v-node , index )
1  if index > total(child)
2   error: "El índice no es válido"
3  else
4   new-op = list( 'v , route-number , node.ID)
5   return (CONCAT ( new-op , NTH_NEIGHBORS(child, index))))
```

Figura 4.6: Función de indexación para la operación seleccionar vehículo de una ruta.

```
NTH_NEIGHBORS( y-node , index )
1 pos = int(index / total(child)) + 1
2 if index == total(child) * (pos - 1)
3 pos = pos -1
4 if pos > y-node.possibilities:
5 error: "El índice no es válido"
6 else
7 new-op = list( 'y , node.ID)
8 new-index = index - (( pos - 1 ) * total(child))
8 return (CONCAT ( new-op , NTH_NEIGHBORS( new-index, child)))
```

Figura 4.7: Funcioón de indexación para la operación seleccionar vehículo de la lista de vehículos disponibles.

- **'v:** hace referencia a la operación de selección de vehículo.

En la figura 4.5 se presenta la implementación de NTH_NEIGHBORS para la selección de vehículo en una ruta. La selección de un vehículo de una ruta solo tiene una variante disponible, es por eso que no es necesario analizar qué variante se escogerá. Simplemente, se agrega esta operación a la lista y se hace un llamado a la misma función, ahora evaluando al nodo hijo.

Si por el contrario, el vehículo se escoge de la lista de vehículos disponibles, la función NTH_NEIGHBORS se define como muestra la figura 4.5. En este caso, se debe analizar todos los posibles vehículos a escoger. Esta función es equivalente en los casos de HFFVRP.

La función de intercambio de vehículos, ya tiene prefijados los valores, por tanto, no aporta a la generación de nuevos vecinos, así que se implementa como muestra la figura

```
NTH_NEIGHBORS( t-node , index )
1  if index > total(child)
2   error: "El índice no es válido"
3  else
4   new-op = list( 't , select-op1.ID , select-op2.ID)
5   return (CONCAT ( new-op , NTH_NEIGHBORS(child, index))))
```

Figura 4.8: Funcioón de indexación para la operación intercambio de vehículos.

Con estas funciones, ya se puede analizar instancias de los problemas HFVRP y HFFVRP en el sistema desarrollado.

4.6. Ejemplos

En esta sección se muestra el funcionamiento de las funciones implementadas. Para ello se define un criterio de vecindad que mezcla las características de HFVRP con las ya implementadas operaciones de CVRP, este se muestra en la figura 4.9. Esta estructura de entorno primero modifica la posición de un cliente dentro de la ruta a la que pertenece y luego modifica el depósito de una ruta.

Se escoge como solución actual la presentada en la figura 4.1 para calcular la cardinalidad de la vecindad resultante.

Se comienza por la primera operación que compone el criterio. Esta operación es principal y en la solución actual se tienen tres rutas posibles, por lo que el árbol de vecindad tiene como raź un **r-node**, con referencia a sus tres hijos. El valor de la cardinalidad de esta operación se calcula como muestra la figura 2.3.

La siguiente operación elemental en el criterio es: Seleccionar cliente c_1 en la ruta r_1. Por lo que cada uno de los hijos de las raíz será un **a-node**. Esta operación es modificadora, esto significa que tendrá un único hijo. El valor de la cardinalidad de esta operación depende de la cantidad de valores que puede tomar la propiedad que la define, como plantea la función presentada en el código de la figura 2.4.

Su hijo será un **b-node**, que será quien represente la tercera operación de la estructura de entorno. La cardinalidad de esta operación depende de la

1. Seleccionar ruta r_1.
2. Seleccionar cliente c_1 en la ruta r_1.
3. Insertar cliente c_1 en la ruta r_1.
4. Seleccionar ruta r_2.
5. Seleccionar vehículo v_1 en la ruta r_2.
6. Seleccionar vehículo v_2 en la lista de vehículos disponibles.
7. Intercambiar v_1 y v_2.

Figura 4.9: Criterio de vecindad

cantidad de valores que puede asumir [11].

La cuarta operación del criterio es una segunda selección de ruta. Esta se calcula de igual forma que la primera.

Sus hijos son representados por **v-node**. Para calcular la cardinalidad de esta operación se sigue el algoritmo presentado en la figura 4.3.

Posteriormente, se selecciona un vehículo dentro de la lista de vehículos disponibles. Esta operación tiene tres valores disponibles, uno por cada vehículo que compone la lista. En ese caso, la cardinalidad se calcula como se muestra en la función de la figura 4.3.

La última operación del criterio es intercambio de vehículos. Esta es una operación pasiva, por lo que se calcula de manera similar a la selección de vehículos dentro de una ruta.

De esta forma, se analiza el criterio, creando el árbol de vecindad que se observa en la figura 4.10.

Se observa que cada nodo tiene asignado un par (x, y), donde las x representan la cantidad de opciones que brinda una operación y las y, la cantidad de vecinos que se generan a partir del subárbol del cual es raíz ese nodo.

A medida que se desciende se actualizan los valores x y se evalúa la función de conteo en los hijos de cada nodo. El resultado serán asignado a los valores y, para terminar el cálculo de la cardinalidad. Al llegar a las hojas estas retornan uno, y en los retornos de la recursividad se reciben los valores calculados de todos los nodos hijos.

Finalmente, en el nodo de la raíz, queda el valor esperado: 126.

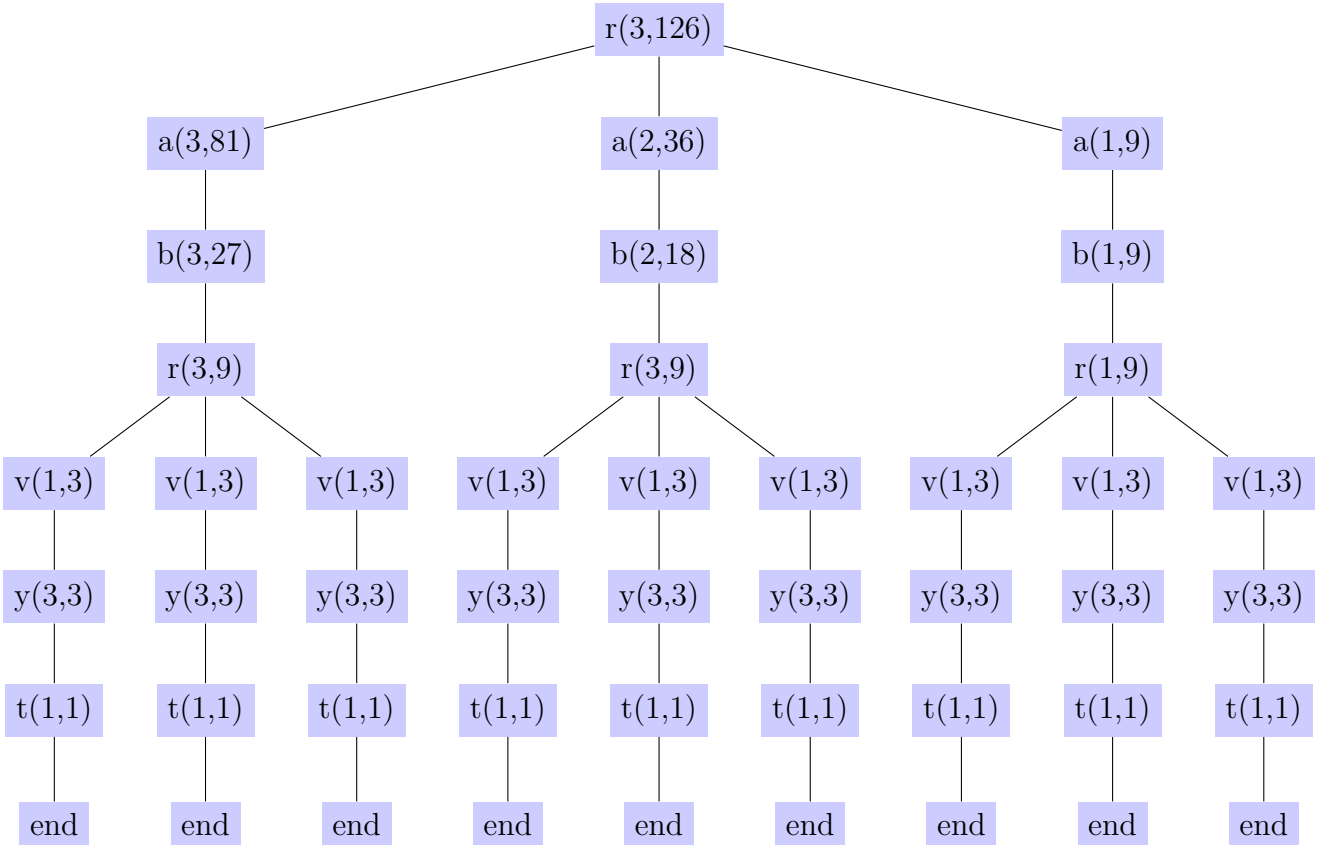

Figura 4.10: Árbol de vecindad expandido para el criterio de vecindad aplicado sobre la solución de la figura 4.1.

Si se desea obtener la solución número ciento veinticuatro de la vecindad se aplica la función NTH_NEIGHBORS comenzando por la raíz, como se explica en [11]. El primer hijo de la raíz contiene 81 soluciones, esto significa que este no será el camino correcto para buscar la solución que se desea. El segundo hijo tiene treinta y seis soluciones, lo que sugiere que hasta él se pueden generar ciento diecisiete soluciones, este camino tampoco es el correcto para encontrar la solución que se busca. El tercer y último hijo genera nueve soluciones, esto quiere decir que bajando por él se llega a la solución buscada. Si el número recibido fuera mayor que la suma de la cantidad de vecinos generados por los tres hijos, entonces el índice no es válido.

Ya se tiene instanciada la primera operación, por lo que la lista de operaciones toma la forma $\{r_1\}$.

En el razonamiento anterior se descartaron ciento diecisiete hijos, esto quiere decir que el siguiente nodo, o sea, el nodo hijo escogido, recibe como índice el valor ocho.

Según [11] la búsqueda continúa para encontrar qué cliente debe ser seleccionado. Para ello, se divide el índice actual entre la cantidad de vecinos generados por el hijo de la operación ($8 \,/\, 9 = 0$) y se aumenta en uno su valor. Esto quiere decir que la operación toma valor uno. La lista de operaciones queda de la siguiente forma $\{r_1,\, a_1 = 1\}$.

El siguiente nodo en el razonamiento es **b-node**. Este nodo se comporta de manera similar a su padre, y también asume valor uno. La lista de operaciones será $\{r_1,\, a_1 = 1,\, b_1 = 1\}$.

La siguiete operación es una selección de ruta. Este nodo se comporta de manera similar a la selección de ruta anterior. Como cada nodo hijo tiene genera tres soluciones y el índice actual es ocho, se escoge el tercer camino, y el índice asume valor dos.

Posteriormente, se tiene un **v-node**. Este nodo representa una operación pasiva. Se calcula su cardinalidad como atendiendo al algoritmo presentado en la figura 4.5 y asume valor uno. La lista de operaciones queda $\{r_1,\, a_1 = 1,\, b_1 = 1,\, r_1,\, v_1 = 1\}$

La siguiente operación en el criterio es selección de vehículo de la lista de vehículos disponibles. La cardinalidad de la misma depende de la cantidad de vehículos disponibles. En este caso, se tienen tres posibilidades. Este nodo genera tres posibles soluciones. Atendiendo al algortimo planteado en la figura 4.5 se divide dos entre el total de soluciones generadas por los hijos y se concluye que la operación toma valor dos. Obteniendo la lista de operaciones $\{r_1,\, a_1 = 1,\, b_1 = 1,\, r_1,\, v_1 = 1,\, v_2 = 2\}$

S = (((2: 1 2 3)
((2: 4 5)
((3: 6))
((1 2 3))

Figura 4.11: Ejemplo de solución MDVRP

La siguiente operación es intercambio de vehículos. Esta operación no aporta nuevos vecinos.

Entonces, la lista de operaciones que debe aplicarse para obtener la solución ciento veinticuatro es $\{r_1,\ a_1 = 1,\ b_1 = 1,\ r_1,\ v_1 = 1,\ v_2 = 2,\ s_1 = 1\}$.

La solución es la mostrada en la figura 4.11.

La naturaleza recursiva de estas implementaciones y las caractrísticas del lenguaje de programación con que se trabaja(LISP), permiten que se creen también criterios donde se combienen las operaciones agregadas para la solución de MDVRP con las de HFVRP. Esto quiere decir que para para criterios como el que se muestra en la figura 4.12 el funcionamiento será análogo al explicado en la presente sección y en la 3.6.

A lo largo de este capítulo se explicó el trabajo realizado para añadir a la herramienta original las operaciones elementales que permiten disenãr estructuras de entorno relacionadas con el problema de enrutamiento de vehículos, cuando la flora es heterogénea. Al finalizar la implementación de las funcionalidades expuestas en este capítulo y el anterior no se pueden declarar en el sistema estructuras de entorno que interactúen con el MDVRP o HFVRP, además, se pueden crear variantes del problema donde ambas restricciones estén presentes gracias al diseño extensible que se empleó para la modelación de las funciones y clases creadas.

1. Seleccionar ruta r_1.
2. Seleccionar cliente c_1 en la ruta r_1.
3. Insertar cliente c_1 en la ruta r_1.
4. Seleccionar ruta r_2.
5. Seleccionar vehículo v_1 en la ruta r_2.
6. Seleccionar vehículo v_2 en la lista de vehículos disponibles.
7. Intercambiar v_1 y v_2.
8. Seleccionar ruta r_3.
9. Seleccionar depósito d_1 en la ruta r_3.
10. Seleccionar depósito d_2 de la lista de depósitos disponibles.
11. Intercambiar d_1 y d_2

Figura 4.12: Criterio de vecindad

Conclusiones y recomendaciones

La presente investigación tributa al sistema creado en la facultad desde 2017, implementando las clases y funciones necesarias para que este sea capaz de explorar vecindades de problemas de enrutamiento de vehículos con múltiples depósitos, flota heterogénea, o la mezcla de ambos. Estas nuevas funcionalidades, unidas a las ya existentes, posibilitan la creación de criterios de vecindad más complejos y adaptables a la vida real.

Para ello, fue necesaria la implementación de funciones de conteo de vecinos que permiten conocer la cardinalidad de la vecindad(COUNT_NEIGHBORS). Esto fue posible gracias a la premisa de que para cada operación se implementa su propia forma de conteo de vecinos y las funcionalidades de Common Lisp que lo permiten.

Además fue necesario agregar las funciones de indexación (NTH_NEIGHBORS) que posibilitan la generación de soluciones nuevas a partir de la actual y un cierto criterio de vecindad.

Se recomienda la implementación de más operaciones elementales que permitan complejizar los criterios de vecindad, como sería la creación de rutas nuevas dentro de los problemas tratados o las necesarias para resolver otras variantes de VRP.

Además, se deben implementar los métodos que calculan la variación en el costo provocada por cada una de esas operaciones.

Bibliografía

[1] A. Mor y M. G. Speranza. Vehicle routing problems over time: a survey. 2018.

[2] Jorge Rodríguez Pérez. Caracterización, modelado y determinación de las rutas de la flota en una empresa de rendering. Master's thesis, 2012.

[3] George B Dantzig and John H Ramser. The truck dispatching problem. *Management science*, 6(1):80–91, 1959.

[4] Geoff Clarke and John W Wright. Scheduling of vehicles from a central depot to a number of delivery points. *Operations research*, 12(4):568–581, 1964.

[5] Claudia Archetti, M Grazia Speranza, and Daniele Vigo. Chapter 10: Vehicle routing problems with profits. In *Vehicle Routing: Problems, Methods, and Applications, Second Edition*, pages 273–297. SIAM, 2014.

[6] Jan Karel Lenstra and AHG Rinnooy Kan. Complexity of vehicle routing and scheduling problems. *Networks*, 11(2):221–227, 1981.

[7] Roberto Baldacci, Paolo Toth, and Daniele Vigo. Exact algorithms for routing problems under vehicle capacity constraints. *Annals of Operations Research*, 175(1):213–245, 2010.

[8] Camila Pérez Mosquera. Primeras aproximaciones a la búsqueda de vecindad infinitamente variable. Master's thesis, Facultad de Matemática y Computación, Universidad de La Habana, La Habana, Cuba, May 2017.

[9] Daniela González Beltrán. *Generación automática de gramáticas para la obtención de infinitos criterios de vecindad en el problema de enrutamiento de vehículos*. PhD thesis, Facultad de Matemática y Computación, Universidad de La Habana, 2019.

[10] José Jorge Rodríguez Salgado. *Una propuesta para la evaluación automática de soluciones vecinas en un Problema de Enrutamiento de Vehículos a partir del grafo de evaluación de una solución*. PhD thesis, Facultad de Matemática y Computación, Universidad de La Habana, 2020.

[11] Héctor Felipe Massón Rosquete. *Exploración de vecindades grandes en el Problema de Enrutamiento de Vehículos usando técnicas estadísticas*. PhD thesis, Facultad de Matemática y Computación, Universidad de La Habana, 2020.

[12] Linda Bibiana Rocha Medina; Elsa Cristina González La Rota and Javier Arturo Orjuela Castro. Una revisión al estado del arte del problema de ruteo de vehículos: Evolución histórica y métodos de solución, 2011.

[13] E. Aarts, J. H. M. Korst, and P. J. M. Van Laarhoven. Simulated annealing. In E. Aarts and J. Lenstra, editors, *Local search in combinatorial optimization*, pages 91–120. John Wiley & Sons, Chichester, 1997.