Universidad de La Habana
Facultad de Matemática y Computación

Extensión del Sistema de Validación para Modelos VRP

Autor:
Carlos Arturo Pérez Cabrera

Tutores:

MSc. Fernando Rodríguez Flores
Lic. Alejandra Monzón Peña

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

Febrero de 2025

github.com/ArturoMetalhead/Extension-del-sistema-de-validacion-para-modelos-VRP

Opinión del tutor

Comprobar si un modelo describe correctamente un VRP es un proceso que puede tomar tiempo, incluso para personas con experiencia en el tema. Desde el año 2020, en nuestra facultad se está desarrollando un sistema para determinar la correctitud de las restricciones de de un modelo de optimización para variantes del VRP. Aunque se han desarrollado varias tesis que tributan a ese fin, aún quedan muchos problemas por resolver. Afortunadamente, y gracias al trabajo de esta tesis, hay dos problemas menos: identificar si un modelo tiene restricciones que garanticen el significado de las variables y construir determinado tipo de soluciones que antes no se consideraban. Durante este trabajo Carlos Arturo trabajó con muchísima independencia, resolviendo los problemas originales que se propusieron y otros que aparecieron durante el camino. Estamos muy contentos con el trabajo realizado y con las nuevas líneas de investigación que de él se derivan. Para el desarrollo de la tesis Carlos Arturo tuvo que incursionar en contenidos y tecnologías que no forman parte de su plan de estudio y lo hizo de una manera excelente. Por todo lo anterior creo que estamos en presencia de un excelente trabajo desarrollado por un excelente científico de la computación.

Febrero, 2025.

MSc. Fernando Raul Rodriguez Flores

Lic. Alejandra Monzón Peña

Resumen

El presente trabajo propone un sistema automatizado para validar modelos de problemas de enrutamiento de vehículos (VRP) en Common Lisp. Dado un modelo matemático y la descripción de un problema de enrutamiento de vehículos, el sistema determina si las restricciones del modelo reflejan adecuadamente las características del problema. Esto es relevante porque la etapa de modelado es un proceso costoso en tiempo humano y propenso a errores. El sistema propuesto ayuda a agilizar la obtención de modelos al automatizar la validación de los mismos. Para lograr este objetivo, se plantean modificaciones a un sistema de validación desarrollado previamente en varias tesis de pregrado de la Facultad de Matemática y Computación de la Universidad de La Habana en los años 2020, y 2023. Las modificaciones propuestas permiten analizar tipos de restricciones que en las versiones anteriores resulta imposible detectar la presencia de errores en las mismas y arregla la generación de las soluciones que se utilizan para comprobar la correctitud de los modelos.

Abstract

The present work proposes an automated system to validate vehicle routing problem (VRP) models in Common Lisp. Given a mathematical model and the description of a vehicle routing problem, the system determines if the model's constraints adequately reflect the problem's characteristics. This is relevant because the modeling stage is a time-consuming and error-prone process. The proposed system helps to accelerate the model acquisition by automating their validation. To achieve this objective, modifications are proposed to a validation system previously developed in several undergraduate theses at the Faculty of Mathematics and Computing of the University of Havana in the years 2020 and 2023. The former modifications allow for the analysis of types of constraints which were impossible to detect in previous versions, the presence of errors in them and fix the generation of solutions used to verify the correctness of the models.

Índice general

Introducción **1**

1. Preliminares **4**
1.1. Fundamentos de los Problemas de Enrutamiento de Vehículos 4
1.2. Variantes de VRPs 5
1.2.1. VRP con Restricciones de Capacidad (CVRP) 5
1.2.2. VRP con Ventanas de Tiempo (VRPTW) 5
1.2.3. VRP con Backhaul (VRPB) 6
1.2.4. VRP con múltiples depósitos (MDVRP) 6
1.2.5. VRP con Entrega y Recogida (VRPPD) 6
1.3. Estrategias de Modelación 7
1.3.1. Modelo a) 7
1.3.2. Modelo b) 9
1.4. LMML 11
1.5. Common Lisp 11
1.5.1. Sintaxis, funciones y macros 12
1.5.2. Sistema de Objetos de Common Lisp (CLOS) 12
1.5.3. Clases 13
1.5.4. Combinaciones de métodos 14

2. Funcionamiento del Sistema **15**
2.1. Descripción del VRP 16
2.2. Generador de Soluciones de Alto Nivel 17
2.3. Generador de Soluciones de Bajo Nivel 20
2.4. Generador de Funciones 22
2.5. Evaluador del Modelo 23
2.6. Implementación en Common Lisp 23
2.7. Limitaciones del sistema 27

3. Propuesta de solución y detalles de implementación **30**
3.1. Propuesta de solución 30
3.1.1. Restricciones de significado 30
3.1.2. Restricciones que nunca se incumplen 35
3.2. Detalles de Implementación 37
3.2.1. Restricciones de significado 37
3.2.2. Restricciones que nunca se incumplen 43

4. Experimentación y resultados **45**
4.1. Modelo 1 con estrategia de Flujo de Mercancias 45
4.1.1. Restricciones 46
4.2. Modelo 2 con estrategia de Flujo de Mercancías 48
4.2.1. Restricciones 49

Conclusiones **51**

Bibliografía **52**

Índice de figuras

1.1. a) Modelo de CVRP utilizando la estrategia de modelación “Flujo de vehículos de dos índices polinomial”. b) Modelo de CVRP utilizando la estrategia de modelación “Flujo de mercancías”. 7

2.1. Arquitectura del sistema de validación automática de modelos de VRP. 15

Introducción

El Problema de Enrutamiento de Vehículos (VRP) es una familia de problemas de optimización combinatoria cuyo objetivo es determinar el conjunto óptimo de rutas que una flota de vehículos debe realizar para atender a un grupo de clientes [22]. Desde la primera aproximación matemática del VRP en 1959 [6] hasta la actualidad, se han desarrollado varios enfoques de solución, tanto con métodos exactos como con heurísticas [22].

La aplicación del VRP a situaciones del mundo real ha demostrado que el uso de procedimientos computarizados para la planificación de la distribución de mercancías logra reducir significativamente los costos de transporte, con ahorros de entre un 5 % y un 20 % de los costos totales [22, 16, 11]. Un paso importante en el proceso de solución de un VRP es la construcción del modelo matemático que lo describe ya que, aunque rara vez se pueden encontrar soluciones exactas debido a la complejidad del problema [13], a partir del modelo se pueden diseñar algoritmos específicos para cada variante [22].

Sin embargo, los modelos que describen los VRP son propensos a errores humanos, y una modelación errónea puede generar soluciones que no cumplan con las características del problema. Esto se debe a la diversidad de VRPs y a la variedad de estrategias de modelación posibles, lo que hace que verificar la correctitud de un modelo sea un proceso complejo. En un estudio de la Facultad de Matemática y Computación del 2020 [21], se propuso un sistema para validar automáticamente modelos de enrutamiento de vehículos para el problema con restricciones de capacidad.

Este sistema utiliza los conceptos de Soluciones de Alto Nivel y Soluciones de Bajo Nivel que, construyéndolas adecuadamente, permiten verificar si las restricciones del modelo reflejan las particularidades del problema. Si en ese sistema se detecta un error en las restricciones del modelo, se puede afirmar que el modelo es incorrecto; en caso contrario, el modelo parece ser correcto [15]. En caso de que el modelo parezca estar correcto no hay garantía de su correctitud, solo se puede afirmar que todas las soluciones factibles que se construyeron satisficieron todas las restricciones y que para toda solución construida de forma que no fuera factible existe al menos una restricción que no se cumple. La implementación del sistema en 2020 [21] demostró la viabilidad de verificar modelos de enrutamiento de vehículos con restricciones de

capacidad, aunque se limitó a una única estrategia de modelación.

Posteriormente, en una tesis de diploma defendida en 2023 [15], se logró extender el sistema propuesto para que pudiera ser aplicado a Problemas de Enrutamiento de Vehículos (VRP) con diversas características, como ventanas de tiempo (VRPTW), recogida y entrega (VRPPD), flota heterogénea, y cualquier otra que se deseen incorporar [15]. Además, el nuevo sistema permite validar modelos utilizando cualquier estrategia de modelación, y no solo una fija.

El sistema original [21], desarrollado en Common Lisp, presentaba limitaciones en la definición de las características del VRP, en la reutilización de clases para almacenar Soluciones de Alto Nivel, y en la representación de estas soluciones como variables y parámetros al agregar nuevas estrategias de modelación. Estas limitaciones fueron abordadas y superadas en la extensión realizada en 2023 [15], lo que permitió una mayor flexibilidad y adaptabilidad del sistema a diferentes tipos de VRP y estrategias de modelación, mejorando significativamente su funcionalidad y usabilidad.

En el presente trabajo se propone una solución para un problema que aún persiste en el sistema: que determinado tipo de restricciones no pueden ser validadas. Actualmente, las restricciones que garantizan que el valor de una variable es el que corresponde con su definición pueden estar incorrectas y el sistema no las detecta.

Por ejemplo, si en un modelo se define la variable u_{ij} como la carga del vehículo al recorrer el arco (i, j), alguna restricción del modelo debe garantizar que al visitar cada cliente i, el valor de esa variable disminuya en la demanda de ese cliente. Tambien se detectó que en la construcción de las soluciones de alto nivel no se considera la posibilidad de que las rutas terminen en el mismo lugar que se comienza y este no sea el depósito, lo que puede ser un error en la implementación de los modelos. Por último, se propone experimentar en busca de errores o posibles mejoras en el sistema de validación automática.

El objetivo general del trabajo es aumentar la usabilidad y versatilidad del sistema desarrollado en la facultad en los últimos años [21, 15], identificar errores en las restricciones que garantizan el significado de las variables y realizar modificaciones para que las rutas puedan terminar en donde mismo se comienza y este no sea el depósito.

La extensión propuesta en el presente documento, permite a los usuarios definir restricciones *de significado* y el sistema verifica si estas restricciones se cumplen. Además que permite la generación de soluciones que terminan en el mismo lugar que se comienza y este no sea el depósito, pues esto es imposible en el sistema anterior debido a la forma en la que se generaban las soluciones. De esta manera se logra una mayor precisión en la validación de los modelos.

Para lograr los objetivos generales se trazaron los siguientes objetivos específicos:

- Estudiar las ideas y códigos del sistema de validación automática para CVRP, con el propósito de determinar qué modificaciones son necesarias.

- Agregar al sistema los elementos necesarios para validar modelos que representen otros problemas existentes.
- Realizar las modificaciones para que las rutas puedan terminar en donde mismo se comienza y este no sea el depósito.
- Experimentar en busca de errores o posibles mejoras en el sistema de validación automática.

Para cumplir los objetivos trazados, se propone considerar dos tipos de restricciones en los modelos: restricciones estructurales que tengan que ver con la forma de las posibles soluciones, y restricciones que garanticen el significado de las variables.

La estructura de este documento es la siguiente. En el capítulo 1, se presentan los preliminares que abarcan los diferentes tipos de Problemas de Enrutamiento de Vehículos (VRP), así como los modelos matemáticos utilizados para describirlos, las estrategias de modelación y algunos aspectos relevantes del lenguaje Common Lisp. El capítulo 2 se centra en la descripción de la herramienta en su estado actual y su funcionamiento. En el capítulo 3, se definen las restricciones de significado explicando cómo estas se integran al sistema para asegurar su correcta validación junto con las modificaciones necesarias para que las rutas puedan terminar en donde mismo se comienza y este no sea el depósito. En el capítulo 4 se ofrecen ejemplos del sistema en acción, ilustrando su aplicación práctica. Por último se presentan las conclusiones, recomendaciones y trabajo futuro.

Capítulo 1

Preliminares

En este trabajo se propone un sistema que valida automáticamente las restricciones de cualquier modelo de optimización relacionado con variantes de Problemas de Enrutamiento de Vehículos (VRPs). Este capítulo introduce los componentes básicos del VRP y algunas de sus variantes. También se detallan las características de Common Lisp, el lenguaje utilizado para implementar el sistema, así como de LMML, el lenguaje de modelación algebraica en el que se deben redactar los modelos a validar con la propuesta de esta tesis.

En la sección 1.1, se describen los fundamentos de los Problemas de Enrutamiento de Vehículos, mientras que la sección 1.2 aborda algunas variantes reportadas en la literatura. La sección 1.3 presenta las estrategias de modelación que se usan para construir los modelos que describan a los VRP. En la sección 1.4 se presenta LMML: Lenguaje de Modelación Matemática en Lisp, que es el lenguaje algebraico elegido para representar el modelo. Finalmente, en la sección 1.5 se presentan las características de Common Lisp, el lenguaje empleado para la implementación del sistema.

A continuación se describen los elementos básicos de los VRP.

1.1. Fundamentos de los Problemas de Enrutamiento de Vehículos

El enrutamiento de vehículos es un desafío crítico en múltiples sectores, ya que juega un papel fundamental en la optimización de operaciones logísticas [22]. Al abordar el problema de enrutamiento de vehículos, las empresas pueden disminuir costos relacionados con el tiempo de entrega, el consumo de combustible y otros gastos operativos, lo que resulta en una mejora significativa en su eficiencia económica. Cualquier otra forma de asignar las rutas es peor que la solución que se obtiene al solucionar el modelo de optimización que representa al VRP y puede producir

pérdidas económicas [11].

En todas las variantes de los VRPs, se pueden identificar tres elementos clave: los clientes que requieren servicios de transporte, los vehículos que transportan las mercancías y los depósitos, desde donde los vehículos inician sus recorridos. Esta estructura básica se mantiene constante, aunque las características específicas de cada variante pueden variar.

La primera formulación del VRP, propuesta en la literatura [6], no consideraba restricciones temporales en las visitas a los clientes, y asumía que todos los vehículos tenían la misma capacidad de carga. Sin embargo, con el tiempo, se han desarrollado variantes que incorporan nuevas restricciones y objetivos. Por ejemplo, algunos modelos incluyen ventanas de tiempo [7], que imponen límites en los horarios de entrega, mientras que otros permiten la entrega y recogida simultánea de mercancías [2], aumentando así la complejidad del problema y la necesidad de soluciones más sofisticadas. En la próxima sección, se presentan algunas de las variantes más relevantes de los VRPs que se reportan en la literatura.

1.2. Variantes de VRPs

En esta sección se analizan algunas de las principales variantes del Problema de Enrutamiento de Vehículos (VRP) como el VRP con restricciones de capacidad (CVRP), el VRP con ventanas de tiempo (VRPTW), el VRP con *backhaul* (VRPB) y el VRP con entrega y recogida (VRPPD) [20]. Cada una de estas variantes puede volverse más compleja al combinarse con otras o al modificar ciertas características.

1.2.1. VRP con Restricciones de Capacidad (CVRP)

El VRP con restricciones de capacidad, conocido como CVRP, es una de las variantes más investigadas en la literatura [6]. Su objetivo es determinar un conjunto óptimo de rutas para una flota de vehículos iguales (flota homogénea) que parte de un depósito central, con la finalidad de abastecer a un grupo de clientes cuyas demandas son conocidas. Cada cliente debe visitarse exactamente una vez, y las demandas no se pueden dividir entre los vehículos. Además, la carga de cada vehículo no debe superar su capacidad máxima. Un ejemplo de CVRP es el caso de una empresa de distribución de alimentos que debe entregar productos a una serie de supermercados en la que cada vehículo tiene una capacidad máxima de carga que no puede ser excedida.

1.2.2. VRP con Ventanas de Tiempo (VRPTW)

El VRP con ventanas de tiempo agrega la restricción de abastecer a los clientes dentro de intervalos de tiempo específicos, además de encontrar rutas óptimas que

cumplan con las restricciones de capacidad [7]. Similar al CVRP, en este caso todos los clientes deben ser visitados una única vez.

La existencia de ventanas de tiempo en cada cliente introduce la necesidad de coordinar las llegadas a los clientes de manera que se respeten sus horarios de disponibilidad. Este problema representa situaciones como el abastecimiento de mercados, que debe hacerse en un horario entre el cierre de la atención al público y la retirada de los empleados a sus hogares.

1.2.3. VRP con Backhaul (VRPB)

En esta variante, el conjunto de clientes se divide en dos subgrupos: El primer grupo incluye los clientes a los que se les debe entregar mercancía de acuerdo a su demanda, mientras que el segundo grupo está compuesto por aquellos cuyas mercancías deben ser recogidas [3].

Al planificar una ruta que incluya ambos tipos de clientes, es obligatorio visitar a los del primer grupo antes de atender a los del segundo. Adicionalmente, la carga de los vehículos no puede exceder en ningún momento su capacidad, y cada cliente debe ser visitado una sola vez. En general, no se permite que una ruta esté compuesta únicamente por clientes del segundo grupo.

1.2.4. VRP con múltiples depósitos (MDVRP)

Esta variante permite que los vehículos salgan de varios depósitos [14]. Un ejemplo representativo sería el modelo de negocios de empresas de comercio electrónico como Amazon [4], que opera con múltiples almacenes para mejorar su capacidad de distribución.

1.2.5. VRP con Entrega y Recogida (VRPPD)

En el VRP con entrega y recogida, a cada cliente se le asignan dos demandas: una para la entrega de mercancías y otra para la recogida [2]. Cada cliente conoce el vértice del que proviene su entrega, y el vértice al que va destinada su recogida. Es esencial que el proceso de entrega se complete antes de que inicie el de recogida. Al igual que en las variantes previamente mencionadas, cada cliente debe ser visitado exactamente una vez, y la capacidad de los vehículos se debe sobrepasar por las cargas. Un ejemplo de VRPPD es la entrega de botellas y recogida de los vacíos.

Los modelos de estos problemas son algunos de los que se desean validar con el trabajo desarrollado en esta tesis, con el fin de verificar si efectivamente reflejan las particularidades de un VRP específico. En la siguiente sección, se explorarán los

modelos matemáticos y las distintas estrategias de modelación que se utilizan para describir problemas de enrutamiento de vehículos.

1.3. Estrategias de Modelación

Para un mismo Problema de Enrutamiento de Vehículos (VRP), existen múltiples modelos que pueden representarlo. La diferencia entre estos modelos radica en el significado de las variables, los parámetros y los conjuntos que utilizan. Un ejemplo de esto se ilustra en la figura 1.1, donde se presentan dos modelos distintos para un mismo VRP.

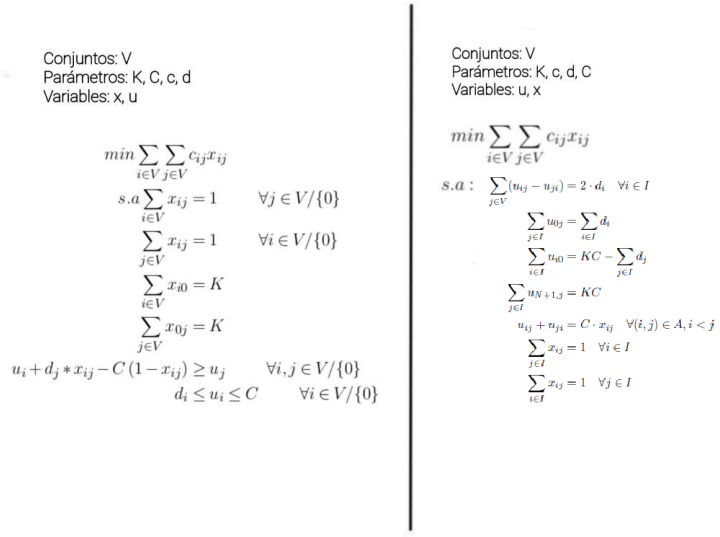

Figura 1.1: a) Modelo de CVRP utilizando la estrategia de modelación "Flujo de vehículos de dos índices polinomial". b) Modelo de CVRP utilizando la estrategia de modelación "Flujo de mercancías".

En el modelo a), se representa el CVRP mediante los siguientes conjuntos, variables y parámetros:

1.3.1. Modelo a)

Conjuntos

- I: Conjunto de clientes $\{1, \ldots, N\}$.
- V: Conjunto que incluye a todos los clientes y al depósito como vértice 0, $\{0, 1, \ldots, N\}$.

Parámetros

- K: Cantidad de vehículos.
- C: Capacidad de los vehículos.
- d_i: Demanda del cliente i.
- c_{ij}: Costo de viajar del cliente i al cliente j.

Variables

- x_{ij}: Variable binaria que indica si el arco (i, j) está incluido en la solución.
- u_i: Variable que representa la carga del vehículo después de visitar al cliente i.

Restricciones

$$\sum_{i \in V} (x_{ij}) = 1, \quad \forall j \in V \setminus \{0\}$$

Esta restricción asegura que cada nodo j (excepto el nodo 0) tenga exactamente un flujo entrante, lo que significa que se debe llegar al cliente j desde un único cliente.

$$\sum_{j \in V} (x_{ij}) = 1, \quad \forall i \in V \setminus \{0\}$$

Similar a la anterior, esta restricción garantiza que de cada nodo i (excepto el nodo 0) se salga exactamente una vez.

$$\sum_{i \in V} (x_{i0}) = K$$

Esta restricción establece que al nodo 0 deben regresar exactamente K vehículos.

$$\sum_{j \in V} (x_{0j}) = K$$

Aquí se indica que del nodo 0 (que representa el depósito) deben salir exactamente K vehículos.

$$u_i + d_j x_{ij} - C(1 - x_{ij}) \geq u_j, \quad \forall i, j \in V \setminus \{0\}$$

Esta restricción garantiza que la variable u_i tenga los valores correctos después de visitar a cada cliente.

$$d_i \leq u_i \leq C, \quad \forall i \in V \setminus \{0\}$$

Estas restricciones establecen límites para el nivel de servicio u_i en cada nodo i. Se asegura que el nivel de servicio no sea menor que la demanda d_i y no exceda una capacidad máxima C. Esto ayuda a mantener un equilibrio entre la demanda y los recursos disponibles.

En el modelo b),se usa la estrategia de modelación Flujo de Mercancías y el CVRP se representa con los siguientes parámetros, variables y conjuntos.

1.3.2. Modelo b)

Conjuntos

- I: Conjunto de clientes $\{1, \ldots, N\}$.
- V: Conjunto que incluye a todos los clientes y al depósito como vértices 0 y $N+1$, $\{0, 1, \ldots, N, N+1\}$.

Parámetros

- K: Cantidad de vehículos.
- C: Capacidad de los vehículos.
- d_i: Demanda del cliente i.
- c_{ij}: Costo de viajar del cliente i al cliente j.

Variables

- x_{ij}: Variable binaria que indica si el arco (i, j) está incluido en la solución.
- u_{ij}: Carga que tiene el vehículo al recorrer el arco (i, j).
- u_{ji}: Carga residual del vehículo al recorrer el arco (i, j).

Restricciones

$$\sum_{j \in V} (u_{ji} - u_{ij}) = 2d_i, \quad \forall i \in I$$

Esta restricción asegura que la diferencia entre el flujo entrante y saliente en cada nodo i (excepto los nodos 0 y n+1) sea igual al doble de la demanda d_i. Esto impone la condición de preservación del flujo.

$$\sum_{j \in I} u_{0j} = \sum_{i \in I} d_i$$

Esta restricción establece que la suma del flujo saliente desde el nodo origen (nodo 0) hacia los nodos j debe ser igual a suma de todas las demandas.

$$\sum_{i \in I} u_{i0} = KC - \sum_{j \in I} d_j$$

Aquí se indica que la suma del flujo entrante al nodo origen desde todos los nodos i debe ser igual a $KC - d_j$, lo que asegura que se respete la capacidad máxima menos la demanda.

$$\sum_{j \in I} u_{n+1,j} = KC$$

Esta restricción establece que el flujo desde el nodo final n+1 debe ser igual a KC, asegurando que se cumpla la capacidad máxima en este punto.

$$u_{ij} + u_{ji} = C x_{ij}, \quad \forall i, j \in A, i < j$$

Esta restricción impone la relación entre las variables de flujo de vehículos. O sea que, como u_{ij} y u_{ji} representan la carga con que el vehículo viaja del cliente i al j y la capacidad residual del vehículo respectivamente, si el arco (i, j) existe entonces la suma de u_{ij} y u_{ji} es igual a la capacidad del vehículo.

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

Esta restricción garantiza que de cada nodo i (excepto los nodos 0 y $(n+1)$) se sale exactamente una vez.

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

Similar a la anterior, esta restricción asegura que a cada nodo j (excepto los nodos 0 y $(n+1)$) se llega exactamente una vez.

La combinación de variables, parámetros y conjuntos que se usan para construir un modelo se denomina estrategia de modelación. Algunas de las estrategias descritas en la literatura [22, 21, 8] incluyen:

- Flujo de mercancías,
- Flujo múltiple de mercancías,
- Flujo de vehículos de dos índices polinomial,
- Flujo de vehículos de dos índices exponencial,

- Flujo de vehículos de tres índices exponencial.

En la figura 1.1, el modelo a) utiliza la estrategia de "Flujo de vehículos de dos índices polinomial", mientras que el modelo b) emplea la estrategia de "Flujo de mercancías".

Dado que el sistema propuesto en este trabajo valida la correctitud de las restricciones de un modelo de VRP, es fundamental que el usuario tenga la capacidad de definir la estrategia de modelación. De no ser así, se restringirían los modelos que podrían ser validados dentro del sistema.

En la siguiente sección se describen las características de LMML, el Lenguaje de Modelación Algebraica en el que se deben representar los modelos a validar.

1.4. LMML

Un modelo de optimización se puede escribir utilizando un Lenguaje de Modelación Algebraica (AML, por sus siglas en inglés). Los AML son lenguajes declarativos de alto nivel que facilitan la creación de modelos de optimización, haciendo el proceso más directo para los programadores y más comprensible para los usuarios [21]. Entre los AML más reconocidos se encuentran GAMS [17], PuLP [1, 19] y Pyomo [10].

En el año 2019, en la Facultad de Matemática y Computación de la Universidad de La Habana, se desarrolló un AML llamado LMML (Lenguaje para la Modelación Matemática en Lisp) que se usa en este trabajo para describir los modelos de optimización que se desean validar.

LMML incluye todas las estructuras fundamentales que se esperan en un AML [5], tales como la declaración de variables, parámetros, conjuntos, funciones y restricciones, también permite el uso de sumatorias y cuantificadores. Además permite representar los modelos en diversos lenguajes de programación. Uno de estos lenguajes es Common Lisp, que en el contexto de este trabajo, permite la evaluación de restricciones matemáticas complejas de manera ágil.

En particular, LMML permite representar las restricciones del modelo como funciones en Common Lisp, y eso se usa en este trabajo para comprobar la validez de dichas restricciones. A continuación, se describen algunas de las características de Common Lisp, el lenguaje de programación en el que se implementó el sistema de validación automática.

1.5. Common Lisp

Common Lisp es un lenguaje de programación multi-paradigma [18]. Esto significa que permite una combinación de paradigmas de programación tales como la

programación imperativa, funcional y orientada a objetos. Facilita el desarrollo de software evolutivo e incremental, con la compilación iterativa de programas eficientes en tiempo de ejecución [18].

A continuación se presenta la sintaxis básica de Lisp y algunas de sus características.

1.5.1. Sintaxis, funciones y macros

La sintaxis de Lisp difiere notablemente de la de lenguajes de programación tradicionales. Dos de sus características más destacadas son el uso extensivo de paréntesis y la notación prefija. Por ejemplo, en lugar de escribir $2 + 3$, en Lisp se expresaría como `(+ 2 3)`.

A los elementos en Lisp se conocen como *s-expresiones*, que son listas de elementos arbitrarios, incluyendo otras listas y átomos. Los átomos son secuencias de caracteres que pueden incluir números, letras y símbolos especiales, mientras que las listas se componen de cualquier cantidad de elementos separados por espacios. Las funciones y macros son *s-expresiones* donde el primer elemento indica el nombre de la función o macro, y los siguientes son los argumentos.

Las macros permiten extender la sintaxis básica del lenguaje a formas más expresivas y adaptadas a problemas específicos [12]. Una macro actúa como una función que toma *s-expresiones* como argumentos y devuelve una *s-expresión* que se evalúa en lugar de la macro. Para facilitar la extensibilidad de la herramienta de validación propuesta en este trabajo, se emplean macros que ofrecen una forma conveniente de definir nuevas características de los VRP. El uso de macros también ayuda a evitar la repetición de código en la implementación de métodos [12].

En el sistema implementado las características de cada variante del VRP se definen mediante clases. La siguiente sección presenta las características del sistema de objetos de Common Lisp.

1.5.2. Sistema de Objetos de Common Lisp (CLOS)

El Sistema de Objetos de Common Lisp (CLOS, por sus siglas en inglés) es un conjunto de operadores que permite implementar programación orientada a objetos en el lenguaje [9]. Este sistema permite definir clases que pueden heredar de múltiples fuentes y está compuesto por funciones genéricas, lo que facilita el uso de polimorfismo con diferentes argumentos y combinaciones de métodos [9]. En las siguientes secciones se describen las principales características de estos elementos.

1.5.3. Clases

Las clases representan los objetos en programación orientada a objetos. En CLOS, una clase se define con un nombre, una lista de clases ancestros y una lista de campos. La lista de clases ancestros especifica las superclases de las que se hereda. Una clase hereda la combinación de campos y comportamientos de sus superclases.

La forma más simple de definir un campo es asignarle un nombre. Sin embargo, en términos generales, un campo se define como una lista que comienza con un nombre seguido por un conjunto de propiedades, como el método de acceso y el valor por defecto. Por ejemplo, para definir la característica de visitar a un cliente una sola vez, se puede definir la siguiente clase:

```
(defclass visit-client-only-once () ())
```

Esta clase no hereda de ninguna superclase y no tiene campos, lo que se indica mediante las dos listas vacías a continuación del nombre, que es equivalente al valor nulo en otros lenguajes de programación.

A diferencia de los sistemas de objetos tradicionales, en CLOS los métodos se definen fuera del cuerpo de la clase mediante funciones genéricas [9]. La siguiente sección describe las principales características de las funciones genéricas.

Funciones genéricas

Una función genérica se compone de uno o más métodos [12]. Los argumentos de cada método pueden especializarse para indicar el tipo de argumentos a los que se aplica. Cuando se invoca una función genérica, se utiliza el método más específico para el cual las clases de los argumentos coinciden con la especialización de los parámetros. Al llamar a una función genérica con argumentos específicos, el orden en que se invocan los métodos varía según la combinación de métodos asociada a esa función genérica. La siguiente sección describe las diferentes variantes para especificar estas combinaciones.

En el siguiente ejemplo se define una jerarquía de clases y métodos para la función genérica `eat` los cuales especializan el parámetro `obj` en cada clase definida:

```
(defclass animal () ())
(defclass dog (animal) ())

(defgeneric eat (obj))
(defmethod eat ((obj animal))
    (format t "Eat like an animal"))
(defmethod eat ((obj dog))
    (format t "Eat like a dog"))
```

A continuación, se presenta cómo se especifican y organizan las combinaciones de métodos, así como su impacto en la invocación de funciones genéricas.

1.5.4. Combinaciones de métodos

Las combinaciones de métodos permiten especificar el resultado de la invocación de una función genérica y el orden en el que deben ejecutarse los métodos de dicha función [12].

Los métodos pueden extenderse mediante métodos auxiliares, incluyendo los métodos *antes de* (`before`) y *después de* (`after`). El método que no tiene ninguna de estas especificaciones se denomina método primario. Los métodos `before` y `after`, como sus nombres indican, se llaman antes y después del método primario, respectivamente.

Al invocar una función genérica, el método más específico es el que tiene los argumentos más especializados dentro de la jerarquía. La combinación estándar de métodos es la siguiente: Se llaman los métodos en el siguiente orden:

1. Todos los métodos `before` (de más a menos específico).
2. El método primario más específico.
3. Todos los métodos `after` (de menos a más específico).

Esto es de vital importancia para definir las restricciones de significado de los modelos de VRP, ya que permite establecer un orden en el que se debe llamar a los métodos que generan las restricciones y especifica cual es el método que se debe ejecutar.

Utilizando los elementos descritos en este capítulo (modelos matemáticos ,las funcionalidades de LMML y Common Lisp), es posible detallar el método propuesto para la validación de modelos de un VRP específico. En el siguiente capítulo se explicará el funcionamiento del sistema de validación.

Capítulo 2

Funcionamiento del Sistema

En 2020, en la Facultad de Matemática y Computación de la Universidad de La Habana se propuso un sistema para la validación automática de modelos para el CVRP [21] y en 2023 se extendió para que pudiera evaluar diferentes tipos de VRPs, no limitándose solo al CVRP [15]. En este trabajo se propone una extensión al trabajo realizado en 2023.

En este capítulo se describe el funcionamiento del sistema de validación automática de modelos de VRP, así como los componentes que lo conforman. Además, se detalla qué limitaciones se encontraron en el sistema original.

La arquitectura del sistema se ilustra en la Figura 2.1, y consta de cuatro componentes principales: el Generador de Soluciones de Alto Nivel, el Generador de Soluciones de Bajo Nivel, el Generador de Funciones y el Evaluador del Modelo. Además, en la Figura 2.1 se muestran las entradas del sistema: la descripción del VRP y el modelo matemático.

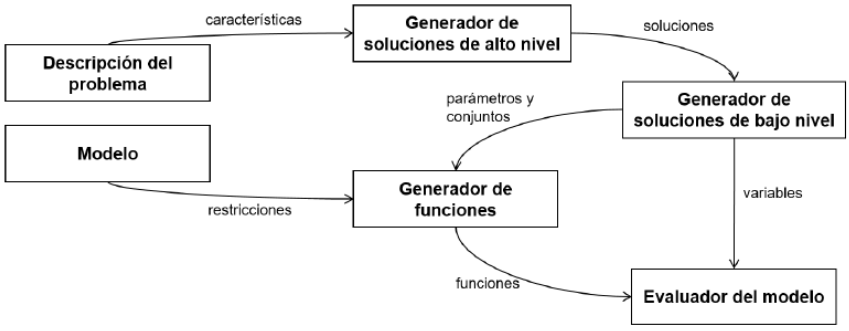

Figura 2.1: Arquitectura del sistema de validación automática de modelos de VRP.

En las siguientes secciones se detalla cómo se representa la descripción del VRP a

partir de un conjunto de características. Además, se explica el funcionamiento de los componentes del sistema: el Generador de Soluciones de Alto Nivel en la sección 2.2, el Generador de Soluciones de Bajo Nivel en la sección 2.3, el Generador de Funciones en la sección 2.4 y el Evaluador del Modelo en la sección 2.5.

2.1. Descripción del VRP

Todos los VRPs tienen una descripción que detalla cómo deben ser las soluciones para este problema. El CVRP se define como un problema en el que se establece una ruta para cada vehículo, cada cliente se debe visitar exactamente una vez y todas las rutas comienzan y terminan en el depósito central. Además, la suma de las demandas de los clientes en cada ruta no debe exceder la capacidad del vehículo [22].

Un VRP cualquiera se puede describir mediante varias características entre las que se encuentran las siguientes:

- Todos los clientes se visitan al menos una vez.
- Cada cliente se visita a lo sumo una vez.
- Los vehículos empiezan su recorrido en el depósito.
- Los vehículos terminan su recorrido en el depósito.
- No sobrepasar la capacidad del vehículo.
- Hay recogida y entrega en los clientes.
- Los clientes tienen ventanas de tiempo.

Por ejemplo, el CVRP se puede describir con las primeras 5 características:

CVRP = {Todos los clientes se visitan al menos una vez,
Cada cliente se visita a lo sumo una vez,
Los vehículos empiezan en el depósito,
Los vehículos terminan en el depósito,
No sobrepasar la capacidad del vehículo}.

Otro ejemplo sería el caso del VRP con ventanas de tiempo (VRPTW). Este VRP consiste en determinar una ruta por cada vehículo, de modo que cada cliente se visita en exactamente una ruta, todas las rutas inician y terminan en el depósito central y la suma de las demandas de los clientes en cada ruta no sobrepasa la capacidad del vehículo. Además, cada cliente debe visitarse en el intervalo de tiempo $[a_i, b_i]$, donde

a_i representa la hora más temprana en que puede iniciar la visita y b_i representa la hora más tardía en que puede hacerse la visita [22]. Tomando en cuenta esto, el VRPTW se puede describir por el conjunto de características:

VRPTW = {Todos los clientes se visitan al menos una vez,
Cada cliente se visita a lo sumo una vez,
Los vehículos empiezan en el depósito,
Los vehículos terminan en el depósito,
Los clientes tienen ventanas de tiempo,
No sobrepasar la capacidad del vehículo}.

Para cada característica se puede definir una característica opuesta.

Por ejemplo, la característica *Todos los clientes se visitan al menos una vez* tiene como opuesto *Al menos un cliente no se visita.*

Definición 2.1 *Una solución es factible para una instancia de VRP si cumple con todas las características de su descripción.*

Si una solución no es factible para una instancia de VRP, es porque incumple al menos una de las características de la descripción. En ese caso, la solución cumple con las características opuestas a las que se incumplen [21].

En la siguiente sección se presentan las Soluciones de Alto Nivel, que en el sistema propuesto se construyen a partir de las características de un VRP.

2.2. Generador de Soluciones de Alto Nivel

Una de las componentes del sistema de validación automática para CVRP [21] es el Generador de Soluciones de Alto Nivel. Este generador crea representaciones de problemas y sus correspondientes soluciones (par problema-solución) para un VRP, considerando elementos como clientes, vehículos y otros componentes relevantes.

Por ejemplo, una Solución de Alto Nivel para el CVRP, con las características mencionadas en la sección 2.1, podría ser:

Depósito: 0,
Clientes: {1, 2, 3},
Demandas: {10, 5, 3},
Capacidad: 14,
Rutas: {(0, 1, 3, 0), (0, 2, 0)}.

En este ejemplo, los datos del problema incluyen el depósito, los clientes, sus demandas y la capacidad de los vehículos. Las rutas representan la solución, y no es necesario especificar el vehículo que recorre cada una, ya que en el CVRP todos los vehículos son iguales y tienen la misma capacidad. El Generador de Soluciones de Alto Nivel utiliza un conjunto de características del VRP para crear estas soluciones.

A continuación se describe el funcionamiento del Generador de Soluciones de Alto Nivel.

El primer paso es construir el conjunto potencia de las características del VRP para obtener todas las combinaciones posibles de características. A continuación, para cada elemento del conjunto potencia se genera una Solución de Alto Nivel que cumpla con las características presentes y con el opuesto de las ausentes.

Para garantizar que cumpla con todas las características en el conjunto y con las opuestas de las que no están, cada solución se construye de forma incremental, agregando información necesaria para cumplir con cada característica [21].

A continuación, se muestra cómo construir una solución de alto nivel que cumpla con todas las características del CVRP:

- No exceder la capacidad de los vehículos,
- Terminar en el depósito,
- Comenzar en el depósito,
- Visitar cada cliente a lo sumo una vez,
- Visitar todos los clientes al menos una vez.

Al invocar al Generador de Soluciones de Alto Nivel, se crea una solución vacía que se modifica para cumplir con cada característica.

Depósito: 0 ,
Clientes: {},
Demandas: {},
Capacidad: 0,
Rutas: {}.

para satisfacer la característica *Visitar todos los clientes al menos una vez*, se agregan los clientes y se crean rutas que los incluyan:

Clientes: {1, 2, 3},
Rutas: {(1, 3), (2)}.

y para *Visitar cada cliente a lo sumo una vez*, se verifica que cada cliente aparezca solo una vez por ruta. En el ejemplo no habría que hacer nada, porque la solución propuesta ya cumple con esa característica.

Para *Comenzar en el depósito*, se agrega el depósito al inicio de cada ruta:

Depósito: 0,
Clientes: {1, 2, 3},
Rutas: {(0, 1, 3), (0, 2)}.

y para *Terminar en el depósito*, se agrega el depósito al final de cada ruta:

Depósito: 0 ,
Clientes: {1, 2, 3} ,
Rutas: {(0, 1, 3, 0), (0, 2, 0)}.

Finalmente, para *No exceder la capacidad de los vehículos*, se agregan las demandas de los clientes y se define la capacidad de los vehículos de forma que no se sobrepase en ninguna ruta:

Depósito: 0 ,
Clientes: {1, 2, 3},
Demandas: {10, 5, 3},
Capacidad: 13,
Rutas: {(0, 1, 3, 0), (0, 2, 0)}.

De esta forma se pueden obtener soluciones que cumplen con todas las características indicadas. Si se especifica una característica opuesta, se obtiene una solución que cumple con la característica opuesta, creando una *Solución de Alto Nivel infactible*. Las soluciones que cumplen con todas las características del VRP se llaman *Soluciones de Alto Nivel factibles*.

Estas *Soluciones de Alto Nivel* son fáciles de generar de forma que satisfagan las características que interesan [15, 21]. Esto permite conocer de antemano, para cada solución, cuándo es factible y cuándo no.

Para comprobar la correctitud de las restricciones solo falta evaluar cada una de esas soluciones en las restricciones del problema. Para evaluarlas en las restricciones, es necesario expresarlas en términos de variables y parámetros del problema, y para eso se usan las soluciones de bajo nivel.

En la siguiente sección se describe cómo se puede usar el *Generador de Soluciones de Bajo Nivel* para convertir una Solución de Alto Nivel a variables, parámetros y conjuntos según la Estrategia de Modelación utilizada en el modelo matemático.

2.3. Generador de Soluciones de Bajo Nivel

Cuando se construye un modelo de optimización para describir un VRP, se utilizan variables y parámetros específicos, de manera que las soluciones del problema se pueden representar mediante estas variables y parámetros.

Por ejemplo, en el modelo para CVRP por Flujo de Vehículos de dos índices polinomial 1.3.1 se tienen las variables x_{ij} y u_i, que indican si el arco (i, j) está en la solución y la carga del vehículo luego de visitar al cliente i respectivamente. Además, se tienen los parámetros: K (cantidad de vehículos), C (capacidad de los vehículos), d_i (demanda del cliente i) y los conjuntos: I (conjunto de clientes) y V (conjunto de clientes incluyendo el depósito como vértice 0).

Si se conoce que las rutas de una solución son:

$$\{(0,1,3,0),(0,2,0)\}.$$

Entonces en la representación de dicha solución con la estrategia de Flujo de dos índices polinomial, se tiene que x_{01}, x_{13}, x_{30}, x_{02} y x_{20} toman valor 1 y las restantes variables x_{ij} toman valor 0.

Para ejemplificar el paso a variables, parámetros y conjuntos, se toma la Solución de Alto Nivel de la sección 2.2:

Depósito: 0 ,
Clientes: {1, 2, 3},
Demandas: {10, 5, 3} ,
Capacidad: 14,
Rutas: {(0, 1, 3, 0), (0, 2, 0)}.

Este ejemplo se puede representar usando la estrategia de modelación Flujo de vehículos de dos índices polinomial de la siguiente forma:

Conjuntos:

$$V : \{0,1,2,3\},$$
$$I : \{1,2,3\}.$$

Parámetros:

$$K : 2,$$
$$C : 14,$$
$$d_i : (10,5,3).$$

Variables:

$$x_{ij} : \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix},$$

$$u_i : \begin{bmatrix} 3 & 0 & 0 \end{bmatrix}.$$

Las variables x_{ij} se representan de forma matricial, donde la posición (i, j) indica el valor de la variable x_{ij}.

El proceso de convertir una Solución de Alto Nivel a variables, parámetros y conjuntos, se realiza en el Generador de Soluciones de Bajo Nivel. El resultado de esta transformación es lo que se denomina Solución de Bajo Nivel y depende de los significados que tengan las variables, parámetros y conjuntos en el modelo matemático. En otras palabras, depende de la estrategia de modelación que se use para construir el modelo.

A continuación se ilustra el proceso con otra estrategia.

Para la estrategia de modelación Flujo de mercancías del CVRP 1.3.2, se tienen las variables x_{ij} y u_{ij}, que indican si el arco (i, j) está en la solución y la carga del vehículo luego de cruzar el arco (i, j) respectivamente. Además, se tienen los parámetros: K (cantidad de vehículos), C (capacidad de los vehículos), d_i (demanda del cliente i) y los conjuntos: I (conjunto de clientes) y V (conjunto de clientes y depósito como vértices 0 y $n+1$).

La misma Solución de Alto Nivel del ejemplo anterior se representa como una Solución de Bajo Nivel en esta estrategia de modelación como:

Conjuntos:

$$V : \{0, 1, 2, 3, 4\},$$

$$I : \{1, 2, 3\}.$$

Parámetros:

$$K : 2,$$

$$C : 14,$$

$$d_i : (10, 5, 3).$$

Variables:

$$x_{ij} : \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix},$$

$$u_{ij} : \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 14 & 0 & 0 & 10 & 0 \\ 14 & 0 & 0 & 0 & 5 \\ 0 & 4 & 0 & 0 & 13 \\ 0 & 0 & 9 & 1 & 0 \end{bmatrix}.$$

Al expresar las Soluciones de Alto Nivel a través de variables, parámetros y conjuntos, se pueden evaluar las restricciones del modelo. Dependiendo de si las restricciones del modelo se cumplen o no tras la sustitución de los valores, se puede determinar la correctitud de las restricciones del modelo matemático.

En el siguiente apartado se explica el funcionamiento del Generador de funciones, un componente del sistema que transforma las restricciones del modelo matemático, descrito en LMML, en funciones de Common Lisp. Las funciones generadas para cada restricción del modelo se evalúan en las Soluciones de Bajo Nivel y, según los resultados, se determina la correctitud del modelo matemático.

2.4. Generador de Funciones

El modelo matemático consiste en un conjunto de ecuaciones e inecuaciones (restricciones) que limitan los valores posibles de las variables para que una solución sea viable. Algunos ejemplos de estas restricciones son:

$$x_{ij} \geq 0, \quad \forall i, j \in V, \quad (2{,}1)$$

$$\sum_{i \in I} x_i = 1. \quad (2{,}2)$$

El propósito del sistema de validación automática es verificar si las restricciones del modelo matemático reflejan las características del VRP, por lo tanto, es necesario determinar si las ecuaciones e inecuaciones del modelo son correctas. Para verificar su correctitud, se evalúan las Soluciones de Bajo Nivel en las restricciones del modelo.

Para realizar esta evaluación, se crea una función para cada restricción que recibe como entrada los valores de las variables, parámetros y conjuntos, y devuelve 1 si la restricción se cumple o 0 en caso contrario.

Por ejemplo, si en una Solución de Bajo Nivel los valores del conjunto I son $\{1, 2, 3\}$ y los valores de las variables x_i son $x_1 = x_2 = 1, x_3 = 0$; la función generada para la restricción (2.2) evalúa 0 en esta Solución de Bajo Nivel.

El Generador de funciones es el componente del sistema que, para cada restricción del modelo, genera una función en Common Lisp que indica si la restricción se cumple o no. Estas funciones se utilizan para evaluar las Soluciones de Bajo Nivel y determinar si las restricciones del modelo son correctas.

En la siguiente sección se describe el Evaluador del modelo que utiliza todos los elementos presentados hasta ahora.

2.5. Evaluador del Modelo

El evaluador del modelo es el componente encargado de la validación y recibe la descripción de un VRP, el modelo matemático en LMML y la estrategia de modelación. A partir de estos datos se generan Soluciones de Alto Nivel que se convierten en Soluciones de Bajo Nivel, las cuales se evalúan en las restricciones del modelo. Utilizando el generador de funciones, se crean funciones en Common Lisp para cada restricción, evaluadas con los valores de las Soluciones de Bajo Nivel.

Para validar las restricciones del modelo, una Solución de Bajo Nivel factible debe cumplir todas las restricciones, mientras que una infactible debe incumplir alguna. Si una Solución de Bajo Nivel factible viola una restricción o una infactible cumple todas, las restricciones son incorrectas. Este es un proceso similar al que realizan los seres humanos para validar un modelo, solo que con la automatización se pueden probar muchos más casos y hacerlo en menor tiempo.

Cuando el sistema detecta que un modelo es incorrecto, se tiene garantía de que alguna de sus restricciones está mal e incluso es posible detectar qué restricciones fallan. En cambio, si no se detectan errores, parece que el modelo está correcto aunque no se tiene garantía de ello.

El sistema se implementó en Common Lisp, y la siguiente sección detalla su implementación.

2.6. Implementación en Common Lisp

La validación automática del sistema se desarrolló utilizando el lenguaje Common Lisp. Para cada característica de los VRPs, se crean dos clases: una que representa la característica y otra que representa su opuesto. Por ejemplo, para la característica *Visitar los clientes al menos una vez*, se definen las siguientes clases:

```
(defclass visit-each-client-at-most-once () ())

(defclass visit-client-more-than-once () ())
```

Los VRP se representan mediante clases que heredan de las características. Por ejemplo:la clase que describe al CVRP hereda de las clases *visitar a los clientes al menos una vez*, *visitar a los clientes a lo sumo una vez*, *empezar en el depósito*, *terminar en el depósito* y *no exceder la capacidad del vehículo.*

```
(defclass cvrp-description (dont-overload-vehicles
                            begin-in-depot
                            end-in-depot
```

```
                              visit-each-client-at-most-once
                              visit-each-client-at-least-once)
  ())
```

Además de la descripción del VRP, el sistema recibe la estrategia de modelación usada para construir el modelo. Para representar la estrategia de modelación se utiliza una clase que contiene campos para almacenar los valores de sus variables, parámetros y conjuntos. Para la estrategia de modelación Flujo de mercancías se tiene la clase:

```
(defclass cvrp-commodity-flow ()
  ((V :accessor V :initarg :V :initform nil)
   (I :accessor I :initarg :I :initform nil)
   (K :accessor K :initarg :K :initform nil)
   (C :accessor C :initarg :C :initform nil)
   (d-i :accessor d-i :initarg :d-i :initform nil)
   (x-ij :accessor x-ij :initarg :x-ij :initform nil)
   (u-ij :accessor u-ij :initarg :u-ij :initform nil)))
```

A partir de la descripción del VRP se construyen las Soluciones de Alto Nivel que se almacenan en una clase `problem-solution` que hay que definir para cada variante de VRP. Para el CVRP se tiene la clase `cvrp-problem-solution` con los campos para almacenar los datos del problema y las rutas de las Soluciones de Alto Nivel que se construyan. A continuación se muestra el código de la clase `cvrp-problem-solution`.

```
(defclass cvrp-problem-solution ()
  ((capacity :accessor capacity :initarg :capacity :initform nil)
   (routes :accessor routes :initarg :routes :initform nil)
   (demands :accessor demands :initarg :demands :initform nil)
   (clients :accessor clients :initarg :clients :initform nil)))
```

Para construir las Soluciones de Alto Nivel se utiliza la función genérica `generate-solution -by-condition`, que recibe una instancia de la clase `problem-solution` para almacenar la Solución de Alto Nivel y una característica (que puede ser también una característica opuesta).

```
(defgeneric generate-solution-by-condition
  (problem-solution characteristic))
```

Para esta función se define un método `after` por cada clase en el sistema que represente una característica o una característica opuesta. Por ejemplo, para la característica *Visitar los clientes al menos una vez* y su opuesto se definen dos métodos de los que se muestran los parámetros que reciben:

```
(defmethod generate-solution-by-condition :after
  (problem-solution
   (characteristic visit-each-client-at-most-once))
  ...)

(defmethod generate-solution-by-condition :after
  (problem-solution
   (characteristic visit-client-more-than-once))
  ...)
```

Para definir qué conjunto de características debe poseer la Solución de Alto Nivel que se construye, se tiene una clase `hls-characteristics` por cada posible combinación de características de la descripción del VRP. En caso del CVRP, dos de las clases `hls-characteristics` que se crean son:

```
(defclass hls-characteristics-1
  (dont-overload-vehicles
   begin-in-depot
   end-in-depot
   visit-each-client-at-most-once
   visit-each-client-at-least-once)
  ())

(defclass hls-characteristics-2
  (dont-overload-vehicles
   begin-in-depot
   end-in-depot
   visit-client-more-than-once
   visit-each-client-at-least-once)
  ())
```

En el ejemplo, la clase `hls-characteristics-1` se forma con todo el conjunto de características del CVRP, mientras que en la clase `hls-characteristics-2` se tienen las características *Empezar en el depósito*, *Terminar en el depósito*, *No exceder la capacidad de los vehículos* y *Visitar a cada cliente al menos una vez*; además se agrega la característica opuesta de *Visitar a cada cliente a lo sumo una vez*.

La función `generate-solution-by-condition` se invoca con una instancia de la clase `problem-solution` y con una instancia de la clase `hls-characterictics`. Por ejemplo:

```
(generate-solution-by-condition
```

```
  (make-instance 'cvrp-problem-solution)
  (make-instance 'hls-characteristics-2))
```

Los métodos de la función `generate-solution-by-condition`, agregan a la Solución de Alto Nivel en construcción, los elementos necesarios para que cumpla con cada característica, como se explicó en la sección 2.2. Para el ejemplo de invocación de la función `generate-solution-by-condition`, se construye una Solución de Alto Nivel infactible para el CVRP, porque cumple con la característica opuesta a *Visitar a cada cliente a lo sumo una vez*.

Cada Solución de Alto Nivel se convierte en una Solución de Bajo Nivel, para esto el Generador de Soluciones de Bajo Nivel tiene las funciones genéricas `generate-variables` y `generate-param-set-values`.

En la función `generate-variables` se asignan los valores de las variables que se usan en el modelo y en la función `generate-param-set-values` se asigna el valor a los parámetros y conjuntos [6]. Ambas funciones tienen como parámetros la clase `problem-solution` y la estrategia de modelación.

Los métodos de las funciones `generate-variables` y `generate-param-set-values` se especializan en la estrategia de modelación. Por ejemplo:

```
(defmethod generate-variables
           (problem-solution
           (model-strategy cvrp-commodity-flow))
  ...)

(defmethod generate-param-set-values
                   (problem-solution
                   (model-strategy cvrp-commodity-flow))
  ...)
```

Al traducir las Soluciones de Alto Nivel a Soluciones de Bajo Nivel, se generan Soluciones de Bajo Nivel tanto factibles como infactibles, las cuales se utilizan para evaluar la correctitud de las restricciones del modelo, como se explicó en la sección 2.3. Para evaluar el modelo, las restricciones se representan como funciones en Common Lisp, permitiendo evaluar los valores de las variables, parámetros y conjuntos de las Soluciones de Bajo Nivel. El proceso de conversión de restricciones en funciones se lleva a cabo en el Generador de funciones, como se explicó en la sección 2.4. Para crear las funciones en Common Lisp, se utiliza el método `generate-functions`, que para una restricción del tipo:

$$\sum_{i \in I} x_i = 1$$

genera el código en Common Lisp correspondiente a la función matemática:

$$f(x, I) = \begin{cases} 1 & \text{si } \sum_{i \in I} x_i = 1 \\ 0 & \text{en otro caso} \end{cases}$$

Esto se hace usando la funcionalidad de LMML que permite describir el modelo en cualquier lenguaje, en este caso funciones de Common Lisp.

En la siguiente sección se abordan algunas limitaciones que presenta el sistema de validación automática de modelos para CVRP [21].

2.7. Limitaciones del sistema

En la versión desarrollada en 2023 [15], el sistema de validación automática de modelos para VRP presenta algunas limitaciones. Una de estas es que el sistema no detecta en las restricciones la falta de restricciones que garanticen que el valor de una variable es el que corresponde con su definición.

Este fenómeno se puede ilustrar usando el siguiente modelo matemático para el CVRP por Flujo de Mercancías:

Conjuntos

- I: Conjunto de clientes $\{1, \ldots, N\}$.
- V: Conjunto con todos los clientes y el depósito como vértices 0 y $N+1$, $\{0, 1, \ldots, N, N+1\}$.

Parámetros

- K: Cantidad de vehículos.
- C: Capacidad de los vehículos.
- d_i: Demanda del cliente i.
- c_{ij}: Costo de ir del cliente i al cliente j.

Variables

- x_{ij}: Variable binaria que representa si el arco (i, j) está en la solución.
- u_{ij}: Carga que tiene el vehículo al recorrer el arco (i, j).

$$\sum_{j \in V} (u_{ij} - u_{ji}) = 2 \cdot d_i \quad \forall i \in I \tag{1.1}$$

$$\sum_{j \in I} u_{0j} = \sum_{i \in I} d_i \tag{1.2}$$

$$\sum_{i \in I} u_{i0} = KC - \sum_{j \in I} d_j \tag{1.3}$$

$$\sum_{j \in I} u_{N+1,j} = KC \tag{1.4}$$

$$u_{ij} + u_{ji} = C \cdot x_{ij} \quad \forall (i,j) \in A, i < j \tag{1.5}$$

$$\sum_{j \in I} x_{ij} = 1 \quad \forall i \in I \tag{1.6}$$

$$\sum_{i \in I} x_{ij} = 1 \quad \forall j \in I \tag{1.7}$$

$$\tag{2.1}$$

La restricción (1.1) impone la condición de preservación del flujo [8]. Las restricciones (1.2), (1.3) y (1.4) garantizan los valores correctos para las variables de flujo de mercancías incidentes en los vértices del depósito. La restricción (1.5) impone la relación entre las variables de flujo de vehículos. Es decir, como u_{ij} y u_{ji} representan la carga con que el vehículo viaja del cliente i al j y la capacidad residual del vehículo respectivamente, si el arco (i, j) está en la solución entonces la suma de u_{ij} y u_{ji} es igual a la capacidad del vehículo. Las restricciones (1.6) y (1.7) definen la obligatoriedad de la visita a cada cliente y de la salida desde el mismo, respectivamente.

En un modelo sin la restricción (1.5), lo que sucede es que esta restricción no representa el cumplimiento de ninguna característica del problema, sino que representa el significado de la variable u_{ij}. La variable u_{ij} de la estrategia de modelación Flujo de Mercancías para CVRP almacena la carga que tiene el vehículo cuando viaja del cliente i al j, mientras que u_{ji} representa la capacidad residual. La restricción (1.5) garantiza que la relación entre u_{ij} y u_{ji} se cumpla, pero, por la forma en que se construyen las soluciones de bajo nivel en el trabajo desarrollado en 2023 [15], resulta imposible que se viole esa restricción , por lo que representa una deficiencia del sistema.

Algo similar ocurre si se elimina del modelo la restricción (1.4), que representa que todos los vehículos terminan sus rutas en el depósito. Para detectar el error en este modelo se tiene que construir una Solución de Alto Nivel infactible que cumpla con las restantes restricciones del modelo. Como esa Solución de Alto Nivel debe cumplir con las restricciones que regulan que cada cliente se visite exactamente una vez y que de todos los clientes se tiene que ir a otro cliente o al depósito, entonces ninguna

ruta puede terminar en un cliente si se desea satisfacer con todas las restricciones excepto la (1.4). Por tanto, todas las rutas tienen que terminar en el depósito, pero mientras que en las Soluciones de Alto Nivel se tiene solamente un depósito; en la representación como variables el depósito inicial (0) se diferencia del depósito final $(N+1)$ por lo que se podrían tener casos en que las rutas terminen en el depósito 0 en vez de en el depósito $N+1$.

Por la forma en que se extraen los valores de las variables binarias x_{ij} que representan si del cliente i se viaja al j, siempre se cambia el depósito del final de las rutas por el $N+1$ y no se construyen nunca Soluciones de Bajo Nivel que terminen en el depósito 0. De esta forma nunca se tendrían en el sistema Soluciones de Bajo Nivel con valores de x_{ij} que satisfagan las restricciones (1.6) y (1.7) pero no la restricción (1.4).

Estos son los problemas principales que se abordan con las modificaciones propuestas en este trabajo. En el próximo capítulo se detallan las modificaciones realizadas al sistema en este trabajo.

Capítulo 3

Propuesta de solución y detalles de implementación

En el capítulo anterior se describió el funcionamiento del sistema de validación automática de modelos para VRP [21]: dada una descripción de un VRP a partir de sus características, se crean las Soluciones de Alto Nivel, que se transforman en Soluciones de Bajo Nivel de acuerdo con la estrategia de modelación que se utilice. Estas Soluciones de Bajo Nivel se evalúan en las restricciones del modelo y se determina la correctitud de estas restricciones.

En este capítulo se presentan las modificaciones realizadas al sistema de validación automática de modelos para VRP y así solucionar las limitaciones presentadas en la sección 2.7: las restricciones con significado y las restricciones que nunca se incumplen.

3.1. Propuesta de solución

En esta sección se presentan las propuestas de solución para las limitaciones presentadas en la sección 2.7.

3.1.1. Restricciones de significado

Al experimentar con el sistema propuesto en 2020 y 2023 se detectó que en ningún caso se comprueba la existencia de restricciones que garanticen que el valor de una variable es el que corresponde con su definición. Esto significa que se puede estar ante un modelo que no representa correctamente las características del problema y el sistema no lo detecte como incorrecto.

Para solucionar este problema se propone generar soluciones de bajo nivel infactibles que satisfagan todas las características del modelo, pero que no cumplan con la

definición de alguna de las variables. A continuación se describe una propuesta para lograrlo.

Definición 3.1 *Se define como restricción de significado a una restricción que garantiza que el valor de una variable es el que corresponde con su definición.*

Por ejemplo, en el modelo 2.7 se tiene la restricción de significado

$$u_{ij} + u_{ji} = C \cdot x_{ij},$$

que garantiza que el valor de las variables u_{ij} y u_{ji} sean siempre una residual de la otra.

Si un conjunto de restricciones no garantiza el significado de una variable, dicha solución infactible cumplirá con todas las restricciones del modelo lo que permitirá deducir que el modelo es incorrecto, como se explicó en la sección 2.5.

La verificación de que las restricciones del modelo garanticen el significado de las variables se realiza antes de comprobar si las restricciones garanticen que una solución cumpla con todas las características del problema.

Definición 3.2 *Se define como función de restricción de significado a un método asociado a una variable que cambia valores de las variables en las soluciones que se van a evaluar para comprobar que se cumpla el significado de la variable asociada al método.*

Ejemplo de esto es la función de restricción de significado para la variable u_{ij} que modificará los valores de las variables, en específico de las x_{ij}, para comprobar que se cumpla con la definición de la variable u_{ij}. Esto se debe a que si se cambian los valores de las x_{ij} pierden sentido los valores de u_{ij} porque deben cumplir con la restricción $u_{ij} + u_{ji} = C \cdot x_{ij}$.

Las variables a las que se les quiere comprobar su significado se registran en una lista con su nombre. Por cada variable que se tenga registrada se crean soluciones de bajo nivel independientes de las soluciones de bajo nivel de las demás variables a las que se les quiere comprobar su significado.

Por ejemplo, si se quiere comprobar el significado de la variable u_{ij} que representa la carga que tiene el vehículo cuando viaja del cliente i al j y de x_{ij} que representa si el vehículo viaja del cliente i al j, se tendrán dos conjuntos de soluciones de bajo nivel, uno para cada variable.

La solución que viola la definición de la variable se crea a partir de una solución de bajo nivel factible del problema.

O sea, se crea una solución de alto nivel que cumpla con todas las características del problema, y esa solución factible para las características se lleva a bajo nivel. Esta

solución de bajo nivel factible se modifica para que incumpla con la definición de la variable.

Las variables de la solución de bajo nivel que se modifican para incumplir con la definición que se está analizando dependen de la variable a la que se le quiere comprobar el significado, pues estas necesitan tener correlación.

En el caso de la variable u_{ij} se modifican los valores de las variables x_{ij} para que la restricción $u_{ij} + u_{ji} = C \cdot x_{ij}$ no se cumpla, pues al tener x_{ij} valores incorrectos va a incumplir con la restricción de significado de la variable u_{ij}. Las variables que se modifican pueden no ser siempre las mismas, pues se podría tener más de una forma de comprobar el significado de una variable. Para la variable u_{ij} puede haber otra forma de modificar las variables con la cual se pueda verificar que se cumpla con el significado de la variable.

Estas modificaciones se realizan de manera controlada para que la solución infactible no incumpla con ninguna otra restricción del modelo, pues debe seguir cumpliendo con las características del problema. La forma en la que se modifican las variables del modelo para comprobar que se viole la definición de una de ellas depende del programador.

A continuación se presenta un ejemplo.

En el modelo 2.7 se tiene la variable u_{ij} que representa la carga que tiene el vehículo cuando viaja del cliente i al j y u_{ji} que representa la capacidad residual del vehículo. Para garantizar que el valor de la variable u_{ij} es el que corresponde con su definición se tiene la restricción $u_{ij} + u_{ji} = C \cdot x_{ij}$.

Para comprobar el significado de la variable u_{ij} se generarán soluciones de bajo nivel factibles que posteriormente se modificarán para que cumplan con todas las restricciones del modelo excepto con la restricción de significado de la variable u_{ij}.

Para el caso en que falte dicha restricción, al evaluar el modelo y no incumplir con ninguna restricción siendo la solución evaluada una solución infactible, se puede deducir que el modelo no es correcto. Por tanto no se garantiza que el valor de la variable u_{ij} sea el que corresponde con su definición.

A continuación se presenta un ejemplo del flujo de trabajo para comprobar el significado de la variable u_{ij} con el modelo 2.7.

Se tiene el siguiente conjunto de restricciones que se desea validar en el que falta la restricción $u_{ij} + u_{ji} = C \cdot x_{ij}$.

$$\sum_{j \in V} (u_{ij} - u_{ji}) = 2 \cdot d_i \quad \forall i \in I \tag{3.1}$$

$$\sum_{j \in I} u_{0j} = \sum_{i \in I} d_i \tag{3.2}$$

$$\sum_{i \in I} u_{i0} = KC - \sum_{j \in I} d_j \tag{3.3}$$

$$\sum_{j \in I} u_{N+1,j} = KC \tag{3.4}$$

$$\sum_{j \in I} x_{ij} = 1 \quad \forall i \in I \tag{3.5}$$

$$\sum_{i \in I} x_{ij} = 1 \quad \forall j \in I \tag{3.6}$$

El conjunto de características de un CVRP es:

```
(defclass cvrp-description (dont-overload-vehicles
                            begin-in-depot
                            end-in-depot
                            visit-each-client-at-most-once
                            visit-each-client-at-least-once)
  ()).
```

Al generarse soluciones de alto nivel factibles se obtienen soluciones como la siguiente:

```
Depósito: 0,
Clientes: {1, 2, 3, 4},
Rutas: {(0, 2, 0) (0, 1, 4, 3, 0)},
Demandas: {0, 81, 62, 75, 65},
Capacidad : 225.
```

Esta se convertirá en una solución de bajo nivel como la siguiente:

```
K: 2,
C: 225,
d_i: (0, 81, 62, 75, 65),
V: {0, 1, 2, 3, 4, 5}.
```

$$u_{ij} = \begin{pmatrix} 0 & 221 & 62 & 0 & 0 & 0 \\ 4 & 0 & 0 & 0 & 140 & 0 \\ 163 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 150 & 0 \\ 0 & 85 & 0 & 75 & 0 & 0 \\ 0 & 0 & 225 & 225 & 0 & 0 \end{pmatrix},$$

$$x_{ij} = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

En este caso, para comprobar el significado de la variable u_{ij} se cambiarán valores para que no se cumpla la restricción $u_{ij} + u_{ji} = C \cdot x_{ij}$, que es la que garantiza el significado de la variable u_{ij}.

El método para garantizar que el valor de la variable u_{ij} es el que corresponde con su definición cambia los valores de la variable x_{ij} para que no se cumpla la restricción de significado de la variable u_{ij} al mismo tiempo que se cumplan con las demás restricciones del modelo.

Los métodos para garantizar los significados de las variables se especializan en cada variable a la que se le quiere comprobar su significado.

En el caso de este método especializado en comprobar el significado de la variable u_{ij}, primero generará rutas nuevas y distintas a las originales que se crearon en el *Generador de Soluciones de Alto Nivel*, que cumplan con las características del problema y luego se generarán los nuevos valores de x_{ij} de la misma forma en la que se generan los originales, o sea, a partir de estas rutas recién creadas.

Estas rutas nuevas no sobreescribirán a las rutas originales en la solución pero la matriz x_{ij} sí reemplaza a la matriz x_{ij} de la solución original, por lo que los valores de los x_{ij} nuevos no corresponderán con los valores de los x_{ij} originales.

Este proceso se realiza de manera controlada para que el cambio de valores no afecten a las demás restricciones del modelo. En el caso de u_{ij} se cambian los valores de x_{ij} teniendo en cuenta las características del problema para que no se incumpla con otras restricciones del modelo.

Las rutas nuevas que se generarán serán como las siguientes:

```
Rutas: {(0, 2, 0) (0, 4, 3, 1, 0)}.
```

La ruta del vehículo 2 es diferente a la original, cumple con las características del problema y, por tanto, con las restricciones del modelo. Entonces se obtendrá una

matriz para x_{ij} que no es la que corresponde con la solución a evaluar, como por ejemplo:

$$x_{ij} = \begin{pmatrix} 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

Se puede observar que los valores de x_{ij} siguen cumpliendo las características del problema, pues se sigue visitando a todos los clientes exactamente una vez y se sale de todos los clientes. La nueva matriz no influye en la capacidad de los vehículos por lo que no se excede la capacidad de los vehículos y se empieza y termina en el depósito puesto que el inicio y final de las rutas es el original por la característica de empezar y terminar en el depósito.

Como los valores de la matriz x_{ij} son distintos a los originales se generarán contradicciones entre los valores de u_{ij} y u_{ji} pues la definición de estas dos variables dependen de los valores de x_{ij}, y por tanto se incumplirá con la restricción de significado de la variable u_{ij} en el caso de que existiese.

En el ejemplo anterior es visible que para la mayoria de las i, j se incumple que $u_{ij} + u_{ji} = C \cdot x_{ij}$, por lo que los valores de u_{ij} no son los que corresponden con su definición. Para el caso $i = 0, j = 1$ se tiene que $u_{01} + u_{10} = 221 + 4 = 225 \neq 0 = 225 \cdot 0 = C \cdot x_{01}$.

Al evaluar el modelo con esta solución de bajo nivel infactible se obtiene que cumple con todas las restricciones del modelo pues en esta solución se sigue cumpliendo con las características del problema. Por tanto como pasa todas las restricciones del modelos siendo una solución infactible se puede deducir que el modelo no es correcto.

A continuación se detalla la propuesta de solución para restricciones que nunca se incumplen.

3.1.2. Restricciones que nunca se incumplen

En el modelo 2.7 se tiene la restricción (1.4) que garantiza que todos los vehículos terminan sus rutas en el depósito. $\sum_{j \in I} u_{N+1,j} = KC$

De la manera en que se crean las soluciones de bajo nivel nunca se van a poder crear soluciones que cumplan con las restricciones $\sum_{j \in I} x_{ij} = 1 \quad \forall i \in I$ (1.6) y $\sum_{i \in I} x_{ij} = 1 \quad \forall j \in I$ (1.7) pero no con la restricción $\sum_{j \in I} u_{N+1,j} = KC$ (1.4) como se mencionó en la sección 2.7.

Cuando se generan las soluciones de alto nivel se tiene solamente un depósito. Sin embargo, en la representación como variables el depósito inicial (0) se diferencia del

depósito final ($N+1$), por lo que se podrían tener casos en que las rutas terminen en el depósito 0 en vez de en el depósito $N+1$.

Por la forma en que se extraen los valores de las variables binarias x_{ij} que representan si del cliente i se viaja al j, siempre se cambia el depósito del final de las rutas por el $N+1$ y no se construyen nunca soluciones de bajo nivel que terminen en el depósito 0. De esta forma nunca se tendrían en el sistema soluciones de bajo nivel con valores de x_{ij} que satisfagan las restricciones (1.6) y (1.7) pero no la restricción (1.4).

Para solucionar este problema se propone que a la hora de generar las soluciones de alto nivel se creen casos en que los vehículos entren y salgan de los clientes a los que vayan a visitar y no terminen en el depósito.

Para esto se modifican las funciones que generan las soluciones de alto nivel para garantizar que, además de las soluciones que ya creaba, se generen otras nuevas soluciones que garanticen lo planteado anteriormente. En específico, las funciones que generan las rutas de los vehículos se modificaron para que genere rutas que terminen en el cliente por el que empezó la ruta y no en el depósito. Si la ruta que se está creando tiene la característica de que empieza y termina en cualquier lugar se generará una ruta que empiece y termine en un cliente creando un subciclo.

Estas nuevas soluciones no están asociadas a nuevas características del problema, sino que se generan como parte de la comprobación de las restricciones del modelo que ya se realizaba en el sistema. Por tanto, se generan en las mismas funciones que generan las soluciones de alto nivel.

Por ejemplo, para los siguientes datos del problema:

```
Depósito: 0
Clientes: {1, 2, 3}
Demandas: {9, 1, 3}
Capacidad: 14
```

En el generador de soluciones de alto nivel se crearán soluciones de alto nivel infactibles que entren y salgan de los clientes y no terminen en el depósito (que se representa por el valor 0) como la siguiente:

```
Rutas: {( 1, 3, 1), (0, 2, 0)}.
```

La ruta del primer vehículo empieza y termina en el cliente 1 creando un subciclo en el que se sale y se entra del cliente 1 y 3 justo como se desea. La ruta del segundo vehículo entra y sale del cliente 2 y del depósito. Por tanto se tiene que la solucion infactible cumple con las restricciones (1.6) y (1.7) pero no con la restricción (1.4).

A continuación se detalla la implementación de las propuestas planteadas en esta sección.

3.2. Detalles de Implementación

En la sección anterior se realizaron propuestas de modificación para el sistema de validación automática [21]. Para agregar al sistema cada una de estas propuestas se implementaron nuevas funciones y clases en Common Lisp. En esta sección se detallan las modificaciones realizadas en el sistema.

3.2.1. Restricciones de significado

Una vez que se hayan generado las soluciones de bajo nivel factibles, se procede modificarlas para comprobar el significado de las variables.

Para cada variable a la que se le quiera comprobar su significado se genera una clase que la representa, la cual es independiente de las clases que forman la estrategia de modelación. Esto se hace con el objetivo de distinguir entre los métodos que se van a ejecutar para comprobar el significado de las variables. Estas clases llevan el nombre de la variable a la que representan y se les añade el sufijo `-meaning`.

Para implementar las restricciones de significado se creó una función genérica `meaning-restriction` a la que se le agregan métodos `after` por cada variable de la que se tiene su función de significado. Este método genérico se especializa en las clases que llevan el nombre de la variable que se desea darle significado para que así la distinción de llamadas de los métodos `after` sea más sencilla, legible y extensible.

Por ejemplo, para la variable `vehicles-charge-residual-from-client-i-to-j` que representa la carga residual de un vehículo al pasar del cliente i al j se tiene la clase `vehicles-charge-residual-from-client-i-to-j-meaning`.

Así, al llamar al método genérico `meaning-restriction` pasándole la clase `vehicles-charge-residual-from-client-i-to-j-meaning` llamará directamente al método asociado a esa clase que es el que modifica los valores para que la comprobación de significado sea correcta.

Los métodos `meaning-restriction` reciben como parámetros una instancia de `problem-solution` que son los datos del problema, una instancia de la clase que representa a la variable a la que se le quiere comprobar el significado, una instancia `variable-parameter` que es donde se guardan las variables,parámetros y conjuntos en el *Generador de Soluciones de Bajo Nivel* y un flujo de salida para mostrar los resultados.

Si se desea extender alguna restricción de significado, o sea, agregar más funcionalidad de la que hacía el método original, solo haría falta crear una clase que herede de la clase asociada a dicha restricción de significado e implementar el método que se va a ejecutar después del método original.

Por ejemplo si se deseara extender la restricción de significado de la variable `vehicles-charge-residual-from-client-i-to-j` se crearía una clase que herede

de la clase `vehicles-charge-residual-from-client-i-to-j-meaning` y se implementaría el método que se va a ejecutar después del método original. Para extender una clase se hace de la siguiente forma:

```
(defclass VEHICLES-CHARGE-RESIDUAL-FROM-CLIENT-I-TO-J-MEANING ()
  ())

(defclass VEHICLES-CHARGE-RESIDUAL-FROM-CLIENT-I-TO-J-MEANING-EXTENDED
    (VEHICLES-CHARGE-RESIDUAL-FROM-CLIENT-I-TO-J-MEANING)
  ())
```

Aquí, `VEHICLES-CHARGE-RESIDUAL-FROM-CLIENT-I-TO-J-MEANING-EXTENDED` hereda de `VEHICLES-CHARGE-RESIDUAL-FROM-CLIENT-I-TO-J-MEANING`, permitiendo que se pueda agregar más funcionalidades.

Para implementar el método que se va a ejecutar después del método original se hace de la siguiente forma:

```
(defmethod meaning-restriction :after (problem-solution
(var-param-meaning vehicles-charge-residual-from-client-i-to-j-meaning-extended)
                       variable-parameter
                       stream)
    ...)
```

Los métodos `after` que se especialicen en esta nueva clase se ejecutarán después de los métodos originales definidos para la clase base, permitiendo extender la funcionalidad sin modificar el comportamiento original.

Para crear nuevas restricciones de significado solo se tendría que agregar un nuevo metodo `after` de la función genérica `meaning-restriction` y una nueva clase que represente la variable a la que se le define el significado. Además se deberá registrar el nombre de la variable en la lista de variables que se desea comprobar su significado la cual lleva por nombre `variables-with-meaning`.

A continuación se presenta un ejemplo de cómo se implementa una restricción de significado para la variable x_{ij}.

Primero se crea la clase que representa a la variable x_{ij}:

```
(defclass VEHICLE-GOES-FROM-CLIENT-I-TO-J-MEANING () ())
```

El método que se ejecutará se define como:

```
(defmethod meaning-restriction :after (problem-solution
(variable-parameter-meaning vehicle-goes-from-client-i-to-j-meaning)
                      variable-parameter
                      stream)
    ...)
```

Para registrar la variable en la lista de variables con significado le agregamos el nombre de la variable a la lista `variables-with-meaning`:

```
(defparameter *variables-with-meaning*
          (list 'vehicles-charge-residual-from-client-i-to-j
                'vehicle-goes-from-client-i-to-j))
```

Para la estrategia de modelación Flujo de Mercancías se tiene la variable u_{ij} que representa la carga que tiene el vehículo al recorrer el arco (i, j). Para declarar que esta es una variable a la que le queremos comprobar el significado se guarda en la lista de variables como el símbolo `vehicles-charge-residual-from-client-i-to-j`. La clase que la representa es como sigue:

```
(defclass VEHICLES-CHARGE-RESIDUAL-FROM-CLIENT-I-TO-J-MEANING () ())
```

El método que se ejecutará es de esta forma:

```
(defmethod meaning-restriction :after (problem-solution
(variable-parameter-meaning vehicles-charge-residual-from-client-i-to-j-meaning)
          variable-parameter
          stream)
    ...)
```

Al comprobar la correctitud de las restricciones de un modelo se comprobará si hay problemas con la definición de las variables que se utilicen y se hayan registrado en la lista de variables con significado.

Primero iterará por todas las variables que conforman el problema, si una de estas se encuentra en la lista de variables con significado entonces se crearán soluciones factibles del problema para posteriormente modificarlas con su respectiva función de significado luego de que se conviertan a soluciones de bajo nivel en los métodos de `vrp-from-high-to-low`.

Al evaluar el modelo con estas soluciones infactibles se podrá determinar si el modelo cumple con las restricciones de significado de las variables. Si las soluciones infactibles generadas de esta forma cumplen con todas las restricciones del modelo se podrá deducir que el modelo no es correcto porque no garantizan el significado de las variables.

A continuación se presenta una visión más detallada del ejemplo mostrado en la sección Propuesta de Solución:

Se tiene el siguiente conjunto de restricciones que se desea validar:

$$\sum_{j \in V} (u_{ij} - u_{ji}) = 2 \cdot d_i \quad \forall i \in I \tag{3.7}$$

$$\sum_{j \in I} u_{0j} = \sum_{i \in I} d_i \tag{3.8}$$

$$\sum_{i \in I} u_{i0} = KC - \sum_{j \in I} d_j \tag{3.9}$$

$$\sum_{j \in I} u_{N+1,j} = KC \tag{3.10}$$

$$\sum_{j \in I} x_{ij} = 1 \quad \forall i \in I \tag{3.11}$$

$$\sum_{i \in I} x_{ij} = 1 \quad \forall j \in I \tag{3.12}$$

El conjunto de características del CVRP es:

```
(defclass cvrp-description (dont-overload-vehicles
                            begin-in-depot
                            end-in-depot
                            visit-each-client-at-most-once
                            visit-each-client-at-least-once)
  ())
```

Antes de generarse las soluciones que determinan la correctitud del modelo sin contar con las restricciones de significado se hacen comprobaciones para saber si las restricciones garantizan el significado de las variables. Para esto se itera por todas las variables del problema y si se encuentra en la lista de variables con significado entonces se comprobará si se cumple con su definición.

```
(defparameter *variables-with-meaning*
(list 'vehicles-charge-residual-from-client-i-to-j))
```

Como en el proceso normal, se crean soluciones factibles del problema con el llamado a la función `Generate-solution-by-condition`, la cual genera soluciones de alto nivel como la siguiente:

```
Depósito: 0,
Clientes: {1, 2, 3, 4},
Rutas: {(0, 2, 0) (0, 1, 4, 3, 0)},
Demandas: {0, 81, 62, 75, 65},
Capacidad : 225.
```

Luego se convertirá en una solución de bajo nivel con el llamado a la función `from-high-to-low`.

```
K: 2,
C: 225,
d_i: (0, 81, 62, 75, 65),
V: {0, 1, 2, 3, 4, 5}.
```

$$u_{ij} = \begin{pmatrix} 0 & 221 & 62 & 0 & 0 & 0 \\ 4 & 0 & 0 & 0 & 140 & 0 \\ 163 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 150 & 0 \\ 0 & 85 & 0 & 75 & 0 & 0 \\ 0 & 0 & 225 & 225 & 0 & 0 \end{pmatrix},$$

$$x_{ij} = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

Después de esto, se llamará al método asociado a la clase que representa a la variable que se le quiere comprobar que cumple con su definición.

En el caso del ejemplo se llamará al método asociado a la clase `vehicles-charge--residual-from-client-i-to-j-meaning` que modificará los valores de las variables para que no se cumpla la restricción de significado de la variable. Más específicamente primero generará rutas nuevas y distintas a las originales que se crearon en el *Generador de Soluciones de Alto Nivel*, que cumplan con las características del problema y luego se generarán los nuevos valores de x_{ij} de la misma forma en la que se generan los originales, o sea, a partir de estas rutas recién creadas. Estas rutas nuevas no sobreescribirán a las rutas originales en la solución al contrario de la matriz x_{ij} nueva que sí va suplantar a la matriz x_{ij} original en la solución a evaluar, por lo que los valores de los x_{ij} nuevos no corresponderán con los valores de los x_{ij} originales.

Para esto se hace el llamado al método genérico `meaning-restriction` que se encarga de llamar al método especializado con la clase que lo representa.

```
(defmethod meaning-restriction :after (problem-solution
(variable-parameter-meaning vehicles-charge-residual-from-client-i-to-j-meaning)
                      variable-parameter
                      stream)
    ...)
```

Como resultado de lo anterior se tienen soluciones de bajo nivel infactibles como la siguiente:

```
K: 2,
C: 225,
d_i: (0, 81, 62, 75, 65),
V: {0, 1, 2, 3, 4, 5}.
```

$$u_{ij} = \begin{pmatrix} 0 & 221 & 62 & 0 & 0 & 0 \\ 4 & 0 & 0 & 0 & 140 & 0 \\ 163 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 150 & 0 \\ 0 & 85 & 0 & 75 & 0 & 0 \\ 0 & 0 & 225 & 225 & 0 & 0 \end{pmatrix},$$

$$x_{ij} = \begin{pmatrix} 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

Esta solución se puede notar que no cumple con la restricción de significado de la variable u_{ij}, pues no se cumple para la mayoría de valores de i, j que $u_{ij} + u_{ji} = C \cdot x_{ij}$. Por ejemplo para $i = 0, j = 1$ se tiene que $u_{01} + u_{10} = 221 + 4 = 225 \neq 0 = 225 \cdot 0 = C \cdot x_{01}$.

Una vez contruídas estas soluciones de bajo nivel infactibles, se evalúan en las restricciones. Para esto se hará el llamado a la función `pass-test` que se encarga de comprobar si la solución cumple con las restricciones del modelo. Si la solución infactible cumple con todas las restricciones del modelo se podrá deducir que el modelo no es correcto. En caso contrario todo parece indicar que el modelo es correcto.

Como en este ejemplo el conjunto de restricciones no garantiza el significado de la variable u_{ij}, al evaluarse la solución infactible no se detectará errores en los valores de las variables y se podrá deducir que el modelo no es correcto.

En la siguiente sección se detalla la implementación de las restricciones que nunca se incumplen.

3.2.2. Restricciones que nunca se incumplen

En el *Generador de Soluciones de Alto Nivel* se modificó el método `generate-solution--by-condition` que modifica las soluciones que terminan su ruta en cualquier lugar. Si la solución que se estaba creando había pasado por el método que empieza la ruta en cualquier lugar, siendo esta la única característica que hace a la solución infactible y luego se pasa por el método que termina la ruta en cualquier lugar, se modificará la solución para que la ruta empiece y termine en el mismo lugar y este no sea el depósito. Esto se sabe gracias a una lista que se le pasa al método en la que el primer elemento indica si la solución es factible o no y el segundo elemento indica si la solución empieza en cualquier lugar siendo esta la única característica que hace a la solución infactible.

De esta manera se asegura que siempre que entre en un cliente también salga de él cumpliéndose así las restricciones (1.6) y (1.7) del modelo 2.7 e incumpliendo la (1.4) pues la solución no termina en el depósito.

$$\sum_{j \in I} u_{N+1,j} = KC \tag{1.4}$$

$$\sum_{j \in I} x_{ij} = 1 \quad \forall i \in I \tag{1.6}$$

$$\sum_{i \in I} x_{ij} = 1 \quad \forall j \in I \tag{1.7}$$

Esto garantiza que en la generación de soluciones de alto nivel se incluyan soluciones para poder comprobar de manera más completa si el modelo es correcto.

A continuación se muestra un ejemplo con los detalles del proceso, usando la siguiente solución de alto nivel:

```
Depósito: 0,
Clientes: {1, 2, 3},
Demandas: {9, 1, 3},
Capacidad: 14.
```

Si el *Generador de Soluciones de Alto Nivel* estaba creando una solución infactible que empieza en cualquier lugar y termina en cualquier lugar se tendrán que generar rutas que no empiecen en 0 y que terminen en un cliente. Lo anterior dicho se lleva a cabo en los métodos `generate-solution-by-condition` especializados para la características de *empezar en cualquier lugar* y *terminar en cualquier lugar*.

Si se pasa por el método que modifica la solución para que termine en cualquier lugar habiendo pasado por el método que modifica la solución para que empiece en cualquier lugar entonces se modificará la solución para que empiece y termine en el

mismo lugar y este no sea el depósito. Los métodos anteriormente mencionados se muestran a continuación:

```
(defmethod generate-solution-by-condition :after
         ((characteristic end-anywhere)
     ...)
    ...)

(defmethod generate-solution-by-condition :after
                         ((characteristic begin-anywhere)
                    ...)
    ...)
```

De esta manera se garantiza que siempre que se entre en un cliente también se salga de él cumpliéndose así las restricciones (1.6) y (1.7) del modelo 2.7 e incumpliendo la (1.4) pues la solución no termina en el depósito. Lo que dará una solución infactible como la siguiente:

```
Rutas: {( 1, 3, 1), (0, 2, 0)}.
```

La ruta del primer vehículo empieza y termina en el cliente 1 creando un subciclo en el que se sale y se entra del cliente 1 y 3 justo como se desea. La ruta del segundo vehículo entra y sale del cliente 2 y del depósito. Por tanto se tiene que la solucion infactible cumple con las restricciones (1.6) y (1.7) pero no con la restricción (1.4).

En el siguiente capítulo se presentan los resultados de la experimentación realizada con al sistema con las modificaciones propuestas en este capítulo.

Capítulo 4

Experimentación y resultados

En este capítulo se presentan los resultados de la experimentación realizada con el sistema de validación automática de modelos para VRP [21] con las modificaciones propuestas en el capítulo anterior. Se utilizaron problemas y modelos de los que se conoce de antemano que son correctos o incorrectos. Los experimentos se realizaron con el CVRP y la estrategia de modelación Flujo de Mercancías.

Para las pruebas que se realizaron, el sistema se configuró a 1000 Soluciones de Alto Nivel por cada posible combinación de características del CVRP. Todos los experimentos se realizaron en una computadora con procesador Intel Core i5-8250U a 1.60 GHz y 12 GB de memoria RAM. Realizar las pruebas con 1000 Soluciones de Alto Nivel por cada posible combinación de características del CVRP demoró aproximadamente 4 segundos.

Para los experimentos se utilizaron los modelos para el CVRP que se presentan en próximas secciones.

4.1. Modelo 1 con estrategia de Flujo de Mercancias

Este es el mismo modelo que se presentó en la sección 1.3.2 del capítulo 1.

4.1.1. Restricciones

$$\sum_{j \in V} (u_{ij} - u_{ji}) = 2 \cdot d_i \quad \forall i \in I \tag{2.1}$$

$$\sum_{j \in I} u_{0j} = \sum_{i \in I} d_i \tag{2.2}$$

$$\sum_{i \in I} u_{i0} = KC - \sum_{j \in I} d_j \tag{2.3}$$

$$\sum_{j \in I} u_{N+1,j} = KC \tag{2.4}$$

$$u_{ij} + u_{ji} = C \cdot x_{ij} \quad \forall (i,j) \in A, i < j \tag{2.5}$$

$$\sum_{j \in I} x_{ij} = 1 \quad \forall i \in I \tag{2.6}$$

$$\sum_{i \in I} x_{ij} = 1 \quad \forall j \in I \tag{2.7}$$

En este modelo se usan los conjuntos I, que representa el conjunto de clientes, y V que representa el conjunto de clientes y el depósito como vértices 0 y $N+1$, $\{0, 1, \ldots, N, N+1\}$. También se tiene como parámetros a K, que representa la cantidad de vehículos, C que representa la capacidad de los vehículos y d_i, que representa la demanda del cliente i. El modelo tiene dos variables: x_{ij} que representa si el arco (i,j) está en la solución y u_{ij} que representa la carga que tiene el vehículo al recorrer el arco (i,j) junto con u_{ji} que representa la carga residual que tiene el vehículo al recorrer el arco (j,i). Los significados de cada restricción están descritos en la sección 1.3.

Las características que se utilizaron para describir el modelo son las siguientes:

- No sobrecargar vehículos.
- Comenzar en el depósito.
- Terminar en el depósito.
- Visitar cada cliente a lo sumo una vez.
- Visitar cada cliente al menos una vez.

Se insertó la variable u_{ij} (`vehicles-charge-residual-from-client-i-to-j`) en la lista de variables con significado para su comprobación.

Se realizaron pruebas con 1000 Soluciones de Alto Nivel por cada posible combinación de características del CVRP con el modelo correcto y no se detectaron problemas en la correctitud del modelo.

Para obtener modelos incorrectos se eliminaron las restricciones (2.4) y (2.5) del modelo.

Primero se eliminó la restricción (2.4) que garantiza que todos los vehículos terminan sus rutas en el depósito con el objetivo de comprobar que el sistema genere soluciones que cumplieran con las restricciones (2.6) y (2.7) y detectara el modelo como incorrecto por la falta de la restricción (2.4).

A continuación se muestra un ejemplo de una solución generada por el sistema que ilustra cómo la falta de la restricción (2.4) hace que el modelo sea incorrecto. Con los siguientes datos generados por el sistema:

```
clients: (1 2 3 4 5)
routes: ((2) (4 5 3) (1))
```

Se generaron las siguientes rutas:

```
routes: ((0 2 0) (4 5 3 4) (0 1 0))
```

La ruta del segundo vehículo no termina en el depósito, lo que hace que el modelo sea incorrecto pero sin incumplir que se entre y se salga de cada cliente.

Como un tercer experimento se eliminó la restricción (2.5) que garantiza la relación entre las variables de flujo de vehículos para verificar que en las pruebas de significado de las variables se detectara que la variable u_{ij} no cumple con su definición.

A continuación se muestra un ejemplo de una solución generada por el sistema que ilustra cómo la falta de la restricción (2.5) hace que el modelo sea incorrecto.

Con los siguientes datos generados por el sistema:

```
clients: (1 2 3 4)
routes: ((0 1 3 2 4 0))
demands: (0 23 48 93 69)
capacity: 250
u_{ij}: (0 233 0 0 0 0)
        (17 0 0 210 0 0)
        (0 0 0 133 69 0)
        (0 40 117 0 0 0)
        (0 0 181 0 0 0)
        (0 0 0 0 250 0)

x_{ij}: (0 1 0 0 0 0)
```

```
(0 0 0 1 0 0)
(0 0 0 0 1 0)
(0 0 1 0 0 0)
(0 0 0 0 0 1)
(0 0 0 0 0 0)
```

Se generaron las siguientes nuevas rutas:

```
routes: ((0 1 0) (0 2 4 3 0))
x_{ij}: (0 1 1 0 0 0)
        (0 0 0 0 0 1)
        (0 0 0 0 1 0)
        (0 0 0 0 0 1)
        (0 0 0 1 0 0)
        (0 0 0 0 0 0)
```

Para $i = 1, j = 3$ se tiene que $u_{13} + u_{31} = 210 + 40 = 250 \neq 0 = 250 \cdot 0 = C \cdot x_{13}$ por lo que es una solución infactible. En la solución generada se sigue cumpliendo con todas las restricciones del modelo, por tanto el modelo es incorrecto debido a la falta de la restricción (2.5).

A continuación se presenta el segundo modelo con la estrategia de modelación Flujo de Mercancías.

4.2. Modelo 2 con estrategia de Flujo de Mercancías

Este modelo no se encuentra en la literatura, sino que es una variación del modelo 1.3.2 del capítulo 1.

4.2.1. Restricciones

$$\sum_{j \in V} x_{ij} = 1 \quad \forall i \in I \tag{3.1}$$

$$\sum_{i \in V} x_{ij} = 1 \quad \forall j \in I \tag{3.2}$$

$$\sum_{i \in V} x_{0i} = K \tag{3.3}$$

$$\sum_{j \in V} x_{j,N+1} = K \tag{3.4}$$

$$u_{ij} + u_{ji} = C \cdot x_{ij} \quad \forall (i,j) \in A, i < j \tag{3.5}$$

$$\sum_{j \in V} (u_{ij} - u_{ji}) = 2 \cdot d_i \quad \forall i \in I \tag{3.6}$$

$$\sum_{i \in I} d_i = \sum_{i \in I} u_{0i} \tag{3.7}$$

Para este modelo se consideran dos conjuntos de clientes I y V. I representa el conjunto de clientes y V representa el conjunto de clientes y el depósito como vértices 0 y $N+1$, $\{0, 1, \ldots, N, N+1\}$.

También se tienen los parámetros K[22] que representa la cantidad de vehículos, C que representa la capacidad de los vehículos y d_i que representa la demanda del cliente i.

Las variables son x_{ij} que representa si el arco (i,j) está en la solución y u_{ij} que representa la carga que tiene el vehículo al recorrer el arco (i,j) junto con u_{ji} que representa la carga residual que tiene el vehículo al recorrer el arco (j,i).

Las restricciones (3.1) y (3.2) definen la obligatoriedad de la visita a cada cliente y de la salida desde el mismo, respectivamente. Las restricciones (3.3) y (3.4) aseguran que salgan y regresen exactamente K vehículos del depósito, respectivamente. La restricción (3.5) impone la relación entre las variables de flujo de vehículos. La restricción (3.6) impone la condición de preservación del flujo. La restricción (3.7) asegura que la suma de las demandas es igual a la suma de las cargas salientes del depósito.

Las características que se utilizaron para describir el modelo son las siguientes:

- No sobrecargar vehículos.
- Comenzar en el depósito.
- Terminar en el depósito.

- Visitar cada cliente a lo sumo una vez.
- Visitar cada cliente al menos una vez.

Como en el modelo anterior, la variable a la que se le debe comprobar su significado es u_{ij}, la cual tiene el mismo significado por lo que se comprobó con la misma función de significado que en el modelo anterior. Se realizaron pruebas con 1000 Soluciones de Alto Nivel por cada posible combinación de características del CVRP con el modelo correcto y no se detectaron problemas en la correctitud del modelo. El tiempo de ejecución fue de aproximadamente 5 segundos.

Para obtener modelos incorrectos se eliminaron las restricciones (3.5), (3.1) y (3.2) del modelo, las cuales garantizan el significado de la variable u_{ij}, la obligatoriedad de la visita a cada cliente y la salida desde el mismo, respectivamente.

Primero se eliminó la restricción (3.5) para comprobar que sea consistente el significado de la variable u_{ij} lo cual resultó en que el modelo era erróneo por la falta de alguna restricción de significado para la variable u_{ij}.

Después se eliminó la restricción (3.1) detectando como erróneo el conjunto de restricciones al igual que con la eliminación de la restricción (3.2). Por tanto los resultados de la experimentación fueron satisfactorios.

Al igual que en los trabajos anteriores, del hecho de no encontrar errores durante el proceso de experimentación no se debe asegurar la inexistencia de futuros fallos en el sistema de validación. Cuando el sistema detecta que un modelo es incorrecto, se tiene garantía de que alguna de sus restricciones está mal e incluso es posible detectar qué restricciones fallan. En cambio, si no se detectan errores, parece que el modelo está correcto aunque no se tiene garantía de ello.

En el siguiente capítulo se presentan las conclusiones del trabajo realizado.

Conclusiones

En este trabajo se modificó el sistema de validación automática de modelos para el VRP [15] de forma que se pueda garantizar que el sistema detecte la falta de restricciones que garanticen que el valor de una variable es el que corresponde con su definición, y la correcta validación de los modelos mediante la generación de soluciones que antes no se podían crear.

Se crearon funciones de significado para las variables, en particular para la variable u_{ij} y así garantizar que el valor de la carga que tiene el vehículo al recorrer el arco (i, j) cumple con su definición. Además se modificaron las funciones que generan soluciones de alto nivel para que se puedan crear soluciones en la que la ruta de los vehículos terminen en el mismo lugar donde se comienza y este no sea el depósito.

Para probar el correcto funcionamiento del sistema tras incorporar las nuevas funcionalidades, se realizaron pruebas con 1000 Soluciones de Alto Nivel por cada posible combinación de características del CVRP con el modelo correcto y no se detectaron problemas en la correctitud del modelo.

A partir del trabajo realizado, se recomienda agregar al sistema los elementos necesarios para validar modelos que representen otros problemas existentes. Una limitación notable es la dependencia del sistema en la representación LMML, lo que puede ser una barrera para los usuarios. Se pudiera agregar al sistema una forma más sencilla de representar el modelo matemático, por ejemplo, utilizar LaTeX o incluso diseñar un sistema visual para escribir los modelos.

Por otra parte, la validación de la correctitud de un modelo para los VRP no es el único paso que toma tiempo en el proceso de búsqueda de una solución: construir los modelos del problema también es costoso. Por esto, se propone como trabajo futuro utilizar el sistema de validación automática como componente de un sistema más grande que genere automáticamente las restricciones de un modelo para un VRP descrito mediante sus características. En este escenario, el sistema implementado en este trabajo puede servir como un evaluador de la correctitud de los posibles modelos.

Futuras investigaciones podrían enfocarse en la integración del sistema con otras herramientas de optimización y en la expansión de su capacidad para manejar diferentes tipos de problemas de VRP.

Bibliografía

[1] Stuart Mitchell et al. «An Introduction to Pulp for Python Programmers». En: *Python Papers Monograph* (2009) (vid. pág. 11).

[2] Mari Arnesen et al. «A Traveling Salesman Problem with Pickups and Deliveries, Time Windows and Draft Limits: Case Study from Chemical Shipping». En: *Computers & Operations Research* 77 (ene. de 2017), págs. 20-31. DOI: 10.1016/j.cor.2016.07.017 (vid. págs. 5, 6).

[3] D.O. Casco, L. Golden y Bruce Wasil. «Vehicle routing with backhauls: Models, algorithms, and case studies». En: *Vehicle Routing: Methods and Studies* (ene. de 1998) (vid. pág. 6).

[4] «Centros de distribución de Amazon: vistazo a su logística en un recorrido virtual». En: (sep. de 2021) (vid. pág. 6).

[5] Claudia Porto Copetillo. «LMML: Lenguaje para la Modelación Matemática en Lisp». En: (2019) (vid. pág. 11).

[6] G. B. Dantzig y J. H. Ramser. «The Truck Dispatching Problem». En: *Management Science* 6.1 (1959), págs. 80-91. ISSN: 00251909, 15265501. URL: http://www.jstor.org/stable/2627477 (visitado 20-12-2024) (vid. págs. 1, 5).

[7] Martin Desrochers, Jacques Desrosiers y M Solomon. «A New Optimization Algorithm for the Vehicle Routing Problem with Time Windows». En: *Operations Research* 40 (abr. de 1992), págs. 342-354. DOI: 10.1287/opre.40.2.342 (vid. págs. 5, 6).

[8] Sheila Artiles Fagundo. «Una propuesta para la modelación automática de problemas de enrutamiento de vehículos». En: (2022) (vid. págs. 10, 28).

[9] P. Graham. *ANSI Common Lisp*. Prentice Hall series in artificial intelligence. Prentice Hall, 1996. ISBN: 9780133708752. URL: https://books.google.com/books?id=p60ZAQAAIAAJ (vid. págs. 12, 13).

[10] William Hart. «Python Optimization Modeling Objects (Pyomo)». En: *The University of Auckland* (2009) (vid. pág. 11).

[11] Geir Hasle, Knut-Andreas Lie y Ewald Quak. *Geometric modelling, numerical simulation, and optimization: Applied mathematics at SINTEF*. Dic. de 2014. ISBN: 978-3-540-68782-5. DOI: 10.1007/978-3-540-68783-2 (vid. págs. 1, 5).

[12] Sonya E. Keene. «Object-Oriented Programming in COMMON LISP: A Programmer's Guide to CLOS». En: *Addison-Wesley Professional* (1989) (vid. págs. 12-14).

[13] Jan Lenstra y A. Kan. «Complexity of vehicle routing and scheduling problems». En: *Networks* 11 (oct. de 2006), págs. 221-227. DOI: 10.1002/net.3230110211 (vid. pág. 1).

[14] Parthana Parthanadee y Rasaratnam Logendran. «Multi-Product Multi-Depot Periodic Distribution Problem». En: (mayo de 2002) (vid. pág. 6).

[15] Alejandra Monzón Peña. «Extensión de un sistema de validación automática para modelos de VRP». En: (2023) (vid. págs. 1, 2, 15, 19, 27, 28, 51).

[16] Jussi Rasku et al. «Automatic Customization Framework for Efficient Vehicle Routing System Deployment». En: mayo de 2018, págs. 105-120. ISBN: 978-3-319-54489-2. DOI: 10.1007/978-3-319-54490-8_8 (vid. pág. 1).

[17] Richard E Rosenthal. «Tutorial on GAMS: A Modeling Language for Optimization». En: *Naval Postgraduate School* (1988) (vid. pág. 11).

[18] Guy L. Steele. «Common Lisp: the language». En: *Elsevier* (1990) (vid. págs. 11, 12).

[19] Michael OSullivan Stuart Mitchell y Iain Dunning. «Pulp: A Linear Programming Toolkit for Python». En: *The University of Auckland* (2011) (vid. pág. 11).

[20] Shi-Yi Tan y Wei-Chang Yeh. «The Vehicle Routing Problem: State-of-the-Art Classification and Review». En: *Applied Sciences* 11 (nov. de 2021), pág. 10295. DOI: 10.3390/app112110295 (vid. pág. 5).

[21] Gabriela Argüelles Terrón. «Validación automática de modelos para problemas de enrutamiento de vehículos». En: (2020) (vid. págs. 1, 2, 10, 11, 15, 17-19, 27, 30, 37, 45).

[22] Paolo Toth y Daniele Vigo. *The Vehicle Routing Problem*. Ed. por Paolo Toth y Daniele Vigo. Society for Industrial y Applied Mathematics, 2002. DOI: 10.1137/1.9780898718515. eprint: https://epubs.siam.org/doi/pdf/10.1137/1.9780898718515. URL: https://epubs.siam.org/doi/abs/10.1137/1.9780898718515 (vid. págs. 1, 4, 10, 16, 17, 49).