UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FISICAS Y MATEMATICAS
DEPARTAMENTO DE INGENIERIA INDUSTRIAL

DESARROLLO DE UN FRAMEWORK PARA EL PROBLEMA DE RUTEO DE VEHÍCULOS

MAURICIO ANDRÉS VÁSQUEZ MORALES

MIEMBROS DE LA COMISIÓN EVALUADORA

Sr. Samuel Varas Guevara, Profesor Guía
Sr. Richard Weber Haas
Sr. Yadran Eterovic Solano
Sr. Eugenio Caldentey Morales

TESIS PARA OPTAR AL GRADO DE MAGÍSTER
EN GESTIÓN DE OPERACIONES

SANTIAGO DE CHILE

2007

A mis padres y mi hermano

AGRADECIMIENTOS

Al profesor Samuel Varas, quien me orientó en cómo enfocar esta tesis y siempre mantuvo la mejor de las disposiciones. A mi familia que me apoyó incondicionalmente dándome ánimo y palabras de aliento. A mis amigos Amanda, David, Beatriz, Enrique, Jerónimo, Marcelo, Óscar y Rodrigo quienes me acompañaron y alentaron a lo largo de los cursos y del desarrollo de este estudio. A Isabel Gac quien me ayudó en la etapa final de la tesis. A Julie por su paciencia y apoyo. Finalmente a Pablo Arancibia quien fue fundamental en el proceso de término de esta tesis.

TABLA DE CONTENIDO

Índice de Ilustraciones

RESUMEN EJECUTIVO

Hoy en día, de los costos de logística de las empresas, más de la mitad corresponden a costos de transporte, siendo uno de los problemas importantes a resolver el del ruteo de vehículos (VRP), que consiste en determinar las mejores rutas para entregar – desde una bodega - productos o servicios a los clientes quienes están dispersos geográficamente. Existen muchos programas comerciales que lo resuelven, pero son de un alto precio, sobre todo para las pymes. Es así que se hace necesario entregar una solución de bajo costo, por ejemplo a través del reuso de componentes de software. Uno de los enfoques más usados son los frameworks, que son una arquitectura de software incompleta que el desarrollador adapta a las necesidades del problema específico.

En este trabajo se desarrolló un framework orientado a objetos para el problema de ruteo de vehículos, a partir de diversos esquemas UML que se implementaron. El mecanismo de desarrollo fue similar al de un software sólo que siempre había que tener en mente que se debía abstraer a un problema VRP lo más genérico posible.

En específico se desarrolló un completo diagrama de clases del problema, que comprende los métodos de resolución del problema. En esta tesis se estudiaron en específico las heurísticas que son el enfoque más difundido. También se desarrolló un mecanismo de mapeo entre métodos de solución y problemas, que permite asociar un problema específico con una heurística específica que lo resuelve.

Para comprobar el funcionamiento del framework se desarrolló un software que lo instanciara. Con este software se realizaron algunas pruebas con problemas aleatorios e instancias conocidas del VRP, obteniendo buenos resultados.

Finalmente se hizo un análisis costo-beneficio que mostró que el reuso de software es una alternativa viable económicamente, en comparación con desarrollar múltiples programas.

Como trabajo futuro queda comprobar que otros desarrolladores puedan usar el framework de manera fácil, y para el dominio que aquí se definió. Por otro lado sería interesante desarrollar frameworks para otros problemas de gestión de operaciones como: asignación de tripulación o ubicación de instalaciones.

1. Introducción

En el mundo de los negocios los costos de transporte constituyen más de la mitad de los costos de logística de las empresas [11]. Esta fracción ha crecido en el último tiempo debido a factores como mayor variabilidad en la demanda de los clientes, búsqueda de calidad total en la entrega de servicios, y por supuesto la competencia derivada de la globalización.

En los últimos 50 años, se empieza a desarrollar la investigación de operaciones, y en específico surgen formas analítico-matemáticas de solucionar el problema de distribución de productos, que tiene su componente más importante en el llamado **Problema de Ruteo de Vehículos** (VRP), que es el problema de determinar las mejores rutas para entregar productos a los clientes dispersos geográficamente. Los primeros algoritmos para resolver el VRP aparecen a mediados de los años 50 [17]. Casi en paralelo, se empiezan a desarrollar rápidamente los computadores, teniendo cada vez mayor poder de cálculo, lo cual permitía hacer implementaciones de estos algoritmos y así poder apoyar a las organizaciones a distribuir sus productos de forma más eficiente, y surgen así los **Sistemas de Ruteo de Vehículos Computarizado** (SRVC).

A medida que pasa el tiempo, la tecnología de los SRVC va evolucionando, y si en un principio eran sólo programas de cálculo para encontrar las rutas más cortas, junto con la evolución de la computación, van incorporando: interfaces gráficas, conexión con bases de datos y sistemas de información geográficos. Por otra parte los algoritmos, se van complejizando para pasar a resolver problemas más reales y no tan idealizados, agregando cada vez más restricciones.

En la actualidad existe una gran diversidad de softwares comerciales [19] para resolver el problema de ruteo de vehículos, pero son muchas las empresas que no tienen acceso a estos softwares por los altos costos que involucra: adquisición, adaptación y operación. Es así que se siguen ocupando técnicas rudimentarias, como son basarse en la experiencia práctica de algún operario quien resuelve el problema usando un “lápiz y un mapa”.

Es por estas razones que se hace necesario entregar herramientas de software a un bajo costo - pero no por ello menos eficientes - para resolver el problema de ruteo de vehículos en las empresas. Estas herramientas no necesariamente debieran innovar en nuevos métodos de solución al problema, sólo debieran hacer síntesis de los mejores métodos de resolución existentes y adecuarlos a las aplicaciones prácticas.

Así surge la necesidad de **desarrollar componentes de software reusables** [3], que cada empresa debiera adaptar a las necesidades de sus problemas específicos de ruteo. Para lograr la reusabilidad del software existen diversos enfoques [24] siendo uno de los más interesantes y prometedores el concepto de **framework**.

Es así que esta tesis estudiará dos conceptos interesantes: resolver el problema de ruteo y scheduling de vehículos, y la reutilización del software a través de frameworks.

Problema de Ruteo de Vehículos

El problema de ruteo de vehículos es el problema de determinar las mejores rutas y/o asignaciones para la entrega/retiro de bienes/servicios a clientes que están distribuidos geográficamente [5].

La decisión que involucra el VRP es asignar un grupo de clientes a un grupo de vehículos y choferes, y secuenciar sus visitas. El objetivo del VRP es entregar un producto/servicio minimizando tiempo/distancia/dinero. Las restricciones son completar las rutas con los recursos disponibles y en los límites de tiempo

impuestos por la jornada de trabajo del chofer, velocidad de viaje y requerimientos del cliente, entre otros.

Ejemplos de VRP existen muchos: entrega de correspondencia, retiro de donaciones, transporte de muestras médicas al laboratorio, abastecimiento de negocios de abarrotes minoristas, abastecimiento de bencineras, vehículos para reparaciones, etc.

Framework

Un framework (FW) es la estructura de un software que debe ser adaptado a la medida, por el programador de la aplicación [26]. Un framework incorpora las partes esenciales y comunes de una familia de aplicaciones de un dominio determinado. Tradicionalmente la construcción de frameworks ha seguido un diseño orientado a objetos, aunque no necesariamente. Un FW consiste de muchas clases que a diferencia de una librería de clases, colaboran según un orden o lógica preestablecida que por un lado la limita en cuanto al tipo de aplicación a desarrollar, al mismo tiempo que hace posible verla como una solución genérica a un conjunto de aplicaciones.

Las ventajas de un framework y en general, de cualquier método de reuso de software, son que permiten desarrollar aplicaciones de calidad a un menor costo y menor tiempo, ya que se pueden reutilizar fácilmente activos como son: algoritmos, análisis de requerimientos, esquemas de base de datos, etc.

En el capítulo 2 se definirán los objetivos generales y específicos de esta tesis, así como el plan de trabajo y los resultados esperados. En el capítulo 3 se estudiarán en profundidad los dos elementos que se esbozaron acá, los frameworks y el problema de ruteo de vehículos. En el capítulo 4 se verán los aspectos que involucra un sistema de ruteo de vehículos (SRVC), en que parte de las empresas está el SRVC y que lo caracteriza. En el capítulo 5 se hace el análisis del framework, que comprende el análisis de requerimientos, el estudio de casos y un primer acercamiento a un SRVC que es el modelo conceptual. En el capítulo 6 se propone un diseño para el SRVC con lo que ello implica: diagrama de actividades, diseño de los hotspots, diagrama de clases y diagrama de secuencia. El capítulo 7 es uno de los más importantes ya que en él se estudian los métodos de solución del problema de ruteo de vehículos y de cómo se pueden desarrollar nuevos métodos de solución que se insertan en nuestro framework. En el capítulo 8 se analiza el framework obtenido en los capítulos anteriores, haciendo pruebas empíricas y teóricas con él. Por otro lado se hace una evaluación costo beneficio de hacer un framework versus construir

software adhoc. Y finalmente en el capitulo 9 se muestran las conclusiones del estudio así como el trabajo futuro de éste.

2. Objetivos

2.1 Objetivo General

Desarrollar un framework orientado a objeto a partir del cual se puedan desarrollar aplicaciones de software que resuelvan el problema de ruteo de vehículos de las pequeñas y medianas empresas, que no tienen acceso a softwares comerciales.

2.2 Objetivos Específicos

- Hacer un estudio exhaustivo del problema de ruteo de vehículos (VRP), sintetizando sus características y los distintos métodos de solución.

- Entregar una solución informática -eficaz y económica a la vez- a los problemas de distribución de las empresas chilenas (principalmente PYMES) que no tienen acceso a costosos software de distribución, ni a paquetes de clase mundial.

- Identificar métricas que puedan demostrar la bondad del framework, en lo que se refiere a la completitud de este, o sea que cubra efectivamente el dominio que se definió a priori.

- Buscar mecanismos para evaluar económicamente los beneficios de desarrollar un framework en lugar de soluciones adhoc.

2.3 Plan de Trabajo

1. Estudio de Bibliografía

Se estudiarán en profundidad los conceptos de Frameworks y Ruteo de Vehículos que son claves para el resto del desarrollo de la tesis.

2. Desarrollo del Framework

Una vez ya estudiado el concepto de framework y el dominio del framework, se debe seguir una metodología de desarrollo, esta podría ser por ejemplo procedural u orientado al objeto (dentro del cual hay varios métodos). En este caso será desarrollado con un mecanismo similar al “Análisis y Diseño orientado a Objetos” que se usa en la ingeniería de software, sólo que en el caso de frameworks no hay que diseñar un software específico sino una arquitectura genérica para todos los problemas del dominio. Una de las

principales características a abstraer/esquematizar en el framework son los métodos de solución.

3. Desarrollo de software

Una vez finalizado el desarrollo del framework, se debe validar, desarrollando a partir de él aplicaciones específicas que resuelvan problemas concretos.

2.4 Resultados Esperados

- Un framework para el problema de ruteo de vehículos, o sea: esquemas explicativos del dominio, clases de objetos, artefactos UML, código fuente, código fuente de aplicaciones y documentación [21] [29]

- Un software que esté en condiciones de ser usado para un problema específico de distribución. Este software –al igual que el framework- será programado en Java.

- Una evaluación de la calidad del framework, tanto en sus ventajas cualitativas como cuantitativas.

3. Metodología

Esta tesis tiene la característica especial que cubre dos áreas medianamente ortogonales que son el “Ruteo de Vehículos” y la “Reusabilidad de Software a través de Frameworks”. Siendo la primera perteneciente al área de la Gestión de Operaciones y la segunda a la Ingeniería de Software.

Para poder validar el enfoque que se está dando al tema de la tesis, primeramente se estudiará qué es un framework [12]: sus características; ventajas y desventajas; diseño [25], implementación y uso.

Luego haremos un estudio del estado del arte del problema de ruteo de vehículos. Cuáles son sus objetivos, decisiones y restricciones [7] [9], su formulación matemática, los tipos de problemas, las características comunes y específicas del problema, los enfoques de solución, etc.

3.1 Frameworks

En los últimos 20 años, mientras los usuarios demandan mayor complejidad y funcionalidad de los software, los desarrolladores tienen que manejar la complejidad propia de la lógica de negocios, así como la complejidad de las

herramientas de programación, con limitados recursos económicos y de tiempo. En este escenario, la reutilización de software se constituye como una de las mejores opciones para disminuir costos. La reutilización de software es el proceso de implementar sistemas de software usando activos de software ya existentes. Estos activos de software se pueden usar en el análisis, el diseño, la implementación y las pruebas. Estos activos pueden ser: algoritmos, esquemas de base de datos, una especificación de requerimientos, bibliotecas, etc.

La reutilización de software no es un fin en si misma sino es importante por los beneficios que conlleva cuando se requiere desarrollar muchas aplicaciones en un mismo dominio: Mejora de la productividad (reduce esfuerzo de desarrollo, reduce tiempo de desarrollo, reduce costos de desarrollo), mejora la calidad de los software, fácil mantención.

Entre los enfoques de reutilización de software están los patrones de diseño y los frameworks, siendo este último uno de los que más éxito ha tenido.

3.1.1 Definición y características

Un framework es una mini-arquitectura reutilizable que provee la estructura genérica y el comportamiento para una familia de abstracciones de software, junto con un contexto formado por metáforas que especifican las colaboraciones y el uso en un dominio dado [18]. Un framework es típicamente

orientado al objeto -aunque podría ser procedural- por lo cual es implementado como un conjunto de clases abstractas que definen la funcionalidad central del framework, y con clases concretas para las características específicas de cada aplicación que se deriva del framework.

El dominio que cubre el framework puede ser cualquiera, para el cual se necesite desarrollar un software: científico, industrial, financiero, etc. Por ejemplo: El manejo de cuentas corrientes de un banco.

Los frameworks deben generar las aplicaciones para un dominio entero, por lo tanto debe haber puntos de flexibilidad que se pueda modificar de acuerdo a los requisitos particulares de la aplicación. A estos puntos se les llama **hotspots**, que son las clases o métodos abstractos que deben ser implementados o puestos en ejecución. Los hotspots son las partes variables de una aplicación a otra del dominio de estudio. Por ejemplo en el caso de las cuentas corrientes, se podría considerar un hotspot a las políticas de manejo de sobregiros.

Por otro lado hay características del framework que son inmutables, que no se pueden (ni deben) modificar en las aplicaciones derivadas. Estos puntos, que constituyen el core del framework, se les conoce como **frozenspots**.

3.1.2 Conceptos y Propiedades de un Framework

3.1.2.1 Estructura de un Framework

Se pueden identificar partes [25] en una aplicación desarrollada a partir de un framework, como se muestra en la Figura 1.

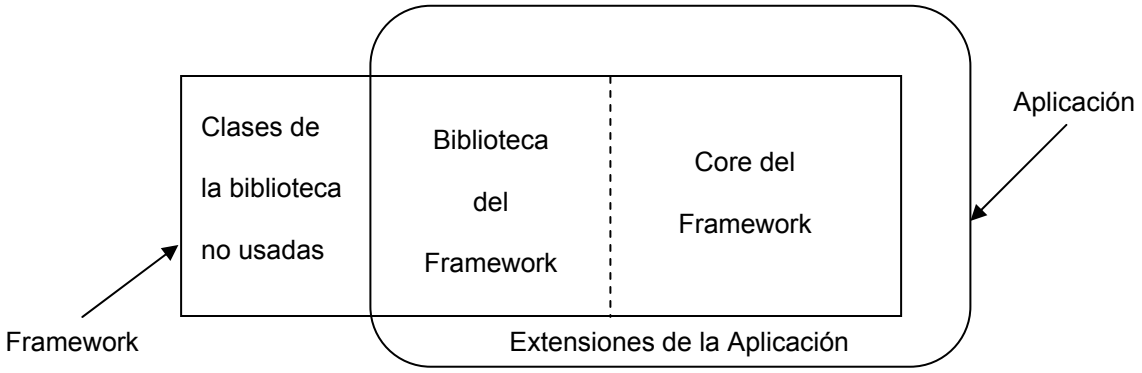


Figura 1: Partes de un Aplicación a partir de un Framework

- Core del Framework: Generalmente consiste de clases abstractas, que definen la estructura genérica, el diagrama de flujo y el comportamiento del framework. Este forma la base de la aplicación desarrollada a partir del framework.
- Biblioteca del Framework: Extensiones del core del framework consiste de componentes concretas que pueden ser usadas con pequeñas o sin modificaciones por las aplicaciones desarrolladas del framework
- Extensiones de la Aplicación: Extensiones especificas de la aplicación hechas al framework.

- Aplicación: En términos del framework, la aplicación consiste del core del framework, de las partes usadas de la bilioteca del framework, y algunas extensiones especificas necesarias para la aplicación
- Clases no usadas de la Librería: Típicamente, no todas las clases de un framework serán necesarias en una aplicación desarrollada a partir de un framework.

3.1.2.2 Tipos de Usuarios

Se pueden identificar tres roles distintos en el desarrollo y uso de frameworks:

- Diseñadores de Framework: Desarrollan el framework original
- Usuarios del Framework: También llamados desarrolladores de aplicaciones, usan (reusan) el framework para desarrollar aplicaciones para problemas específicos del dominio.
- Mantenedores del Framework: Refinan y redesarrollan el framework para cumplir con nuevos requerimientos que hayan surgido.

3.1.2.3 Tipos de Frameworks

Una de las formas en que se puede clasificar a los frameworks es según la forma en que se customizan, o sea, la forma en que se puede desarrollar aplicaciones a partir de él. Existen tres tipos de frameworks [31]:

Frameworks de Caja Blanca: La instanciación del framework es posible a través de la creación de nuevas clases. Estas clases y el código correspondiente se pueden introducir por herencia o composición. Se agregan nuevas funcionalidades creando una subclase de una clase que ya existe en el framework. Para usar frameworks de caja blanca el desarrollador de aplicaciones debe conocer muy bien cómo funciona el framework.

Frameworks de Caja Negra: Producen instancias usando scripts de configuración del framework, con los cuales se configura la aplicación final. Tienen la ventaja que no se requiere que el desarrollador de aplicaciones conozca los detalles internos del framework, por lo cual son mucho más fáciles de usar.

Frameworks de Caja Gris: La mayoría de los frameworks son de Caja Gris, que son aquellos que contienen elementos de Caja Blanca y Caja Negra, ya

que algunas partes se implementan vía herencia o composición, y otras a través de configuración de parámetros.

3.1.2.4 Propiedades deseables de un framework

Los frameworks son hechos para ser usados en aplicaciones específicas, por lo cual lo más importante que debe cumplirse es la reusabilidad, que significa que las ideas, el diseño y el código son desarrollados una vez, y luego usados para resolver muchos problemas, consiguiendo así productividad, confiabilidad y calidad. Un buen framework debe tener ciertas propiedades que lo hacen más reusable [14].

Fácil de Usar. Se refiere a la capacidad que tiene el desarrollador de aplicaciones para usar el framework, el cual debe ser comprensible y debe facilitar el desarrollo de aplicaciones. Los frameworks son para ser reusados, pero aún uno muy bien diseñado, puede no ser usado si es difícil de entender. Para ayudar a la comprensión, el framework debe ser documentado con la descripción de los hotspots, ejemplos de aplicaciones y ejemplos, que el desarrollador pueda usar.

Extensibilidad. Si nuevos componentes o propiedades pueden ser agregados a un framework fácilmente, se dice que este es extensible. Un framework

aunque sea fácil de usar, debe ser extensible. Un framework puede contar con una lista definida y acotada de características, y puede ser muy fácil de comprender y usar, pero su reusabilidad es mayor si es fácilmente extensible para incluir nuevas operaciones.

Flexibilidad. Es la capacidad de usar el framework en más de un contexto. Esto se refiere a la cobertura de dominio del framework. Los frameworks que pueden ser usados en múltiples dominios, como por ejemplo un framework de interfaz de usuario, son especialmente flexibles. Si un framework es aplicable a un dominio amplio, o a varios dominios, entonces este será reusado por más desarrolladores. Sin embargo, la flexibilidad debe ser balanceada con la facilidad de uso. En general, un framework con muchos hotspots será flexible, pero también será difícil de comprender, requiriendo mucho trabajo de parte del desarrollador de aplicaciones para usarlo.

Completitud. La completitud se refiere a que el framework cubra todas los posibles problemas/aplicaciones del dominio especificado. Aunque los frameworks generalmente sean incompletos, ya que no cubren todas las posibles variaciones de un dominio, cierta completitud es una propiedad deseable. Actualmente no existen mecanismos que permitan saber el grado de completitud de un framework.

Consistencia. Debe haber convenciones respecto a los conceptos y definiciones que se usan a lo largo del framework. Se debe usar nombres consistentemente en todo el framework, para facilitar la comprensión de los desarrolladores y ayudarlos a reducir errores en su uso.

3.1.3 Desarrollo de un framework

En general un desarrollo orientado a objeto, y en particular el desarrollo de un framework requiere un enfoque iterativo o cíclico, en el cual el framework es definido, testeado y refinado un número de veces [14]. Las metodologías usuales de diseño orientado a objeto no son suficientes para los frameworks, ya que difieren al menos en dos aspectos. Primero el nivel de abstracción es diferente, ya que los frameworks proveen una solución para un dominio entero de problemas, mientras que las aplicaciones resuelven un problema concreto. Segundo, los frameworks son por naturaleza incompletos, a diferencia de una aplicación, lo que genera complicaciones, por ejemplo, en la etapa del testeo, ya que un framework no se puede probar directamente, sino que a través de aplicaciones que se derivan de él. En la Figura 2 se puede apreciar la diferencia entre desarrollar múltiples aplicaciones adhoc y desarrollar un framework a partir del cual se instancian aplicaciones [26].

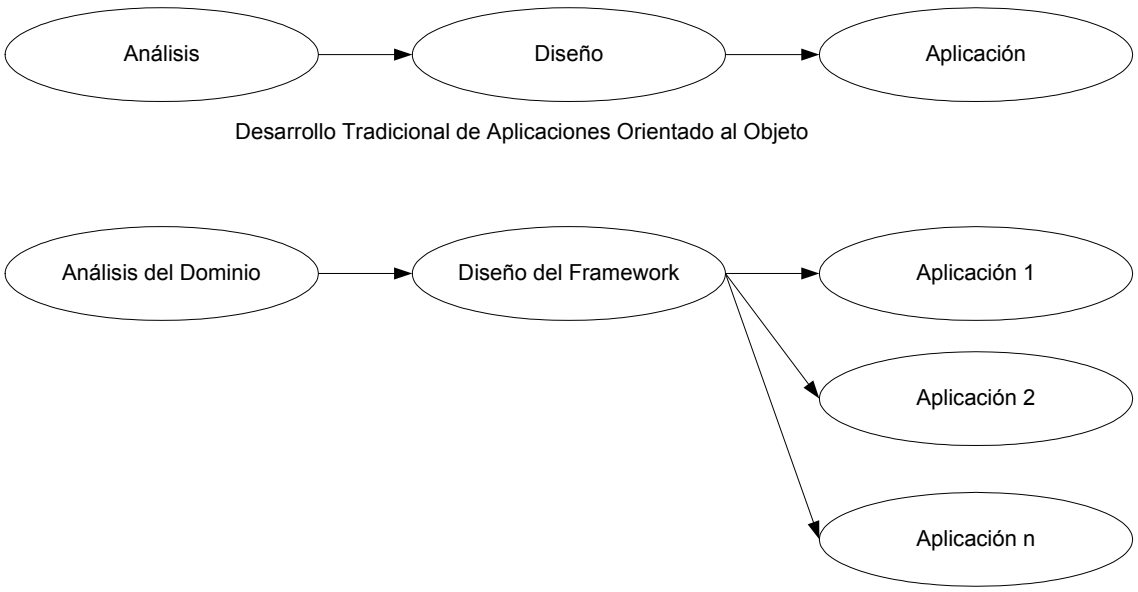


Figura 2: Desarrollo de Aplicaciones basado en Frameworks

Dado que los frameworks son un concepto medianamente nuevo, aún no hay estándares de cómo deben ser construídos, existiendo diversas técnicas que difieren en la forma de cómo se abstrae el diseño. Hay mecanismos bottom-up, top-down, híbridos, por generalización de hotspots, etc. [14]. Pero todas estas formas de diseñar un framework tienen en común tres etapas principales: análisis del dominio, diseño e implementación, instanciación y testeo.

3.1.3.1 Análisis del dominio

Análisis del dominio es el proceso por el cual la información (ver detalle más adelante) usada en desarrollar sistemas de software es identificada, capturada y organizada con el propósito de hacerla reusable cuando se creen nuevas aplicaciones. La persona encargada de analizar el dominio, debe ser generalmente un experto en el dominio, ya que más que nadie conoce los requerimientos de información.

Para analizar el dominio se estudia literatura técnica, software existentes, estándares conocidos, consejos de expertos (en caso de no serlo), estudios de usuarios/clientes, los cuales permiten conocer/determinar el contexto, alcance y límites del dominio. Esta última es una de las decisiones más importantes en el diseño de un framework, ya que determinará cuan flexible será el framework, lo que influirá directamente en el nivel de abstracción que se debe lograr, y por lo tanto, que tan complejo será el desarrollo. También es interesante obtener una taxonomía del problema, un glosario/diccionario de datos, y un modelo funcional del problema.

En esta etapa pueden ayudar herramientas de modelamiento como UML, que es un estándar para el modelamiento orientado a objeto.

3.1.3.2 Diseño e Implementación

El diseño determina la estructura de las abstracciones: frozenspots y hotspots. El diseño e implementación del framework están frecuentemente entrelazados. Las abstracciones pueden ser difícil de diseñar apropiadamente la primera vez y partes del framework deben ser rediseñadas y reimplementadas. Partes del framework pueden sufrir rediseño mientras otras partes están siendo implementadas.

Existen técnicas especificas que ayudan a la etapa de diseño del desarrollo de los frameworks, teniendo en común algunos consejos:

- Reducir el número de clases y métodos que el usuario debe sobrescribir.
- Simplificar la interacción entre el framework y las extensiones que hace la aplicación.
- Aislar el código dependiente de la plataforma.
- Consolidar funcionalidades similares en una sola abstracción.
- Romper grandes abstracciones en pequeñas con gran flexibilidad.
- Usar composición más que herencia.

3.1.3.3 Instanciación y Testeo

La instanciación se refiere a desarrollar aplicaciones a partir del framework, por medio de la implementación de los hotspots, resultando así una aplicación completa, que es ejecutable, a diferencia del framework que no lo es.

Hay dos formas de testear un framework. Primero, debiera ser testeado aisladamente, esto es, sin alguna aplicación que lo instancie. Testear por si mismo el framework ayuda a identificar defectos, ya que se pueden separar los defectos, de los errores que pueden ser causados por una aplicación. Es importante detectar los defectos en un framework, ya que estos serán traspasados a todas las aplicaciones desarrolladas a partir de él.

Segundo, el verdadero test de un framework ocurre sólo cuando este es usado para desarrollar aplicaciones. Instanciar el framework ayuda a exponer áreas donde el framework está incompleto y ayuda a mostrar áreas donde el framework necesita ser más flexible o fácil de usar. Las aplicaciones producidas por este tipo de testeo pueden ser parte de la documentación del framework, ya que indican cómo este se usa.

3.1.4 Documentación de un Framework

La documentación de un framework [29] puede ayudar en todos los aspectos de su uso. Ya que los creadores del framework no estarán siempre disponibles, la documentación pasa a ser el elemento clave, a través del cual los usuarios (desarrolladores de aplicaciones específicas) pueden aprender sobre él.

Los comentarios del código y la documentación son parte necesaria de cualquier proyecto de programación, pero estos son especialmente necesarios cuando se está desarrollando/usando un framework. Si los desarrolladores de aplicaciones no comprenden el framework, simplemente no lo usarán. Por eso se debe proveer bastante información para que los desarrolladores comprendan como usarlo, y así puedan conseguir la solución que ellos buscan.

Hay que dejar claro qué clases pueden ser usadas directamente, qué clases pueden ser instanciadas y qué clases deben ser sobrescritas. Los desarrolladores están interesados en resolver su problema particular, por lo que no son tan importantes los detalles de la implementación del framework.

Dado que no hay un estándar de la documentación a entregar, lo mejor es proveer una variedad de esta [4]:

- Programas de ejemplo
- Descripciones del framework
- Descripciones generales de cómo usar el framework
- Diagramas de la arquitectura del framework
- Artefactos UML del análisis y diseño
- Libros de Recetas que explican como instanciar cada uno de los hotspot
- Manual de Referencia, en el cual se describe cada una de las clases

3.2 El Problema de Ruteo de Vehículos (VRP)

El problema de ruteo de vehículos (VRP) es uno de los problemas clásicos de la investigación de operaciones [7] [8], tanto por su complejidad matemática, ya que es un problema de optimización combinatorial, como por su importancia en la práctica, puesto que está presente en todas las organizaciones que deben entregar un servicio o producto a sus clientes en su domicilio.

El problema de ruteo de vehículos (VRP) es un nombre genérico dado a un conjunto de problemas en los cuales se debe atender la demanda de los

clientes (dispersos geográficamente) por productos o servicios, para lo cual se tiene una flota de vehículos y una o más bodegas desde donde parten y llegan los vehículos de la flota. La solución del problema debe especificar cuales clientes serán servidos por cada vehículo y en que orden para minimizar el costo total sujeto a una variedad de restricciones como capacidad del vehículo y de tiempos de despacho.

3.2.1 Problema Clásico

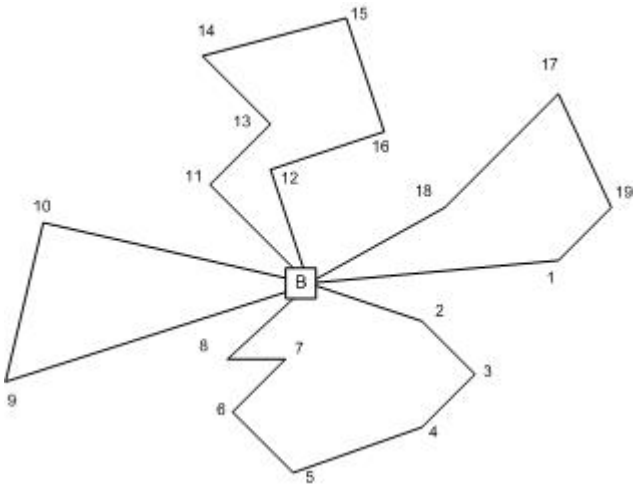
En el más sencillo y clásico de los VRP [7] se tiene una flota de vehículos idénticos, que hacen despachos a los clientes desde una bodega central. El modelo es definido por los siguientes parámetros:

K = número de vehículos de la flota
N = número de clientes a los cuales se les debe despachar. Están indexados del 1 al n, y 0 es la bodega
B = capacidad (por ej. peso o volumen) de cada vehículo
a_i = tamaño del despacho del cliente i, medido en la misma unidad que la capacidad del vehículo
c_{ij} = costo de viaje directo entre los puntos i y j (se supone $c_{ij} >= 0$ y $c_{ij} = c_{ji}$)

El problema es determinar K rutas, donde una ruta es un tour que empieza en la bodega, atiende a un conjunto de clientes en una determinada secuencia y

vuelve a la bodega. Cada cliente debe ser asignado a exactamente a una de las K rutas/vehículos y el total de los despachos no debe exceder la capacidad de vehículo B. Las rutas deben ser elegidas tal que minimicen el costo total de viaje, el cual puede ser la suma de las distancias recorridas, o la suma de los tiempos de viajes.

En la Figura 3 se puede ver un ejemplo de un problema de ruteo de vehículos clásico.



Solución

Vehículo 1: 1 – 19 – 17 – 18

Vehículo 2: 12 - 16 - 15 - 14 - 13 - 11

Vehículo 3: 10 – 9 - 8

Vehículo 4: 8 - 7 - 6 - 5 - 4 - 3 - 2

Figura 3: VRP Clásico

O sea, luego de la resolución del problema se determinó que la primera ruta visitaba primero al cliente 1, luego al 19, luego al 17, finalmente al 18 y vuelve a la bodega; y así con las otras tres rutas.

En el Anexo 1, se puede ver el modelo matemático detrás del problema clásico de VRP.

3.2.2 Clasificación de los VRP

Los VRP son una familia de problemas en los que se tiene clientes con demandas, y una flota que partiendo desde bodega(s) los recorren distribuyendo/recogiendo productos. Pero existen muchas variaciones de estos, las cuales es conveniente estudiar con detalle, para obtener mejores soluciones para cada uno de los tipos de problemas.

Primero clasifiquemos los tipos de servicios a los clientes, estos pueden ser:

i. Sólo servicio: No se entrega ni recoge ningún producto. Por ejemplo: Servicio Técnico.
ii. Despacho o recolección pura: Donde a cada cliente se le entregan/reciben productos. Por ejemplo: Comida a domicilio o Recolección de basura.
iii. Despacho primero, Recolección Después: Primero se entregan productos a ciertos clientes y luego se recorre a los que entregarán productos.

iv. Despacho y Recolección Combinada: Algunos clientes entregan productos y otros reciben. Por ejemplo: Un servicio de correo rural donde los clientes reciben paquetes y también pueden enviar encomiendas.

Otra forma de clasificar los problemas de ruteo es de acuerdo a la importancia de los aspectos geográfico y temporales [7]:

i. Ruteo de Vehículos: En este tipo de problemas no hay restricciones de tiempo, sólo deben obtenerse las rutas para cada vehículo y los clientes que deben visitar, sin interesar cuando. Por ejemplo: Repartición del diario en las mañanas.
ii. Scheduling de Vehículos: Se debe visitar a cada cliente en una hora predefinida, y el servicio en sí, también demora cierto tiempo. También es considerado un problema de scheduling si es que hay restricciones de precendencia, o sea que un determinado cliente debe ser atendido antes que otro.
iii. Combinado: Son igualmente importantes los aspectos temporales como los geográficos. La mayoría de los problemas en la vida real son de este tipo.

También se pueden clasificar, de acuerdo al lugar donde se realiza el servicio, que típicamente es en los nodos de una red, pero también existen problemas

donde el servicio tiene lugar en los arcos como por ejemplo en el barrido de las calles o la recolección de basura.

Además de estas 3 formas de clasificar los problemas, se pueden caracterizar los problemas de ruteo con una lista más detallada de sus características [6]. Ahora se presentan 10 características en las cuales los problemas difieren.

1. Tamaño de la Flota
    - 1 vehículo
    - más de 1 vehículo
2. Tipo de Flota
    - Homogénea (todos los vehículos iguales)
    - Heterogénea (no todos iguales)
3. Naturaleza de la demanda
    - Determinística
    - Estocástica
4. Ubicación de la Demanda
    - En los nodos (no necesariamente todos)
    - En los arcos (no necesariamente todos)
    - Mezclados
5. Red
    - No direccionada
    - Direccionada
    - Mezclada
6. Restricciones a la capacidad del vehículo
    - Impuestas - todas iguales
    - Impuestas – no todas iguales

- Sin restricciones

7. Máximo Vehículo Ruteo Time
    - Impuestas - todas iguales
    - Impuestas – no todas iguales
    - Sin restricciones
8. Costos
    - Variables o Costo de Ruteo
    - Fijo operacional o costo adquisición vehículo
9. Operación
    - Despachos solamente
    - Recolección solamente
    - Ambos
10. Objetivo
    - Minimizar costos de ruteo
    - Minimizar suma de costos fijos y variable
    - Minimizar el número de vehículos requeridos

Tomando distintas combinaciones de opciones, de cada una de las 10 características, se puede obtener un gran número de posibles escenarios. Por ejemplo el problema del Vendedor Viajero (TSP) tiene una bodega, un vehículo de capacidad infinita, demandas determinísticas que necesitan ser servidas en los nodos de una red no direccionada, sin restricciones de tiempo a la ruta, y el objetivo es minimizar la distancia viajada.

3.2.3 Otras Restricciones

A pesar que hemos clasificado los problemas de ruteo, igual en los problemas del mundo real surgen muchas otras restricciones, que complican la formulación y solución del problema:

- Ventanas de Tiempo. El cliente puede especificar que desea ser atendido en un rango de horas.
- Restricciones de precedencia pueden imponer un orden parcial a la secuencia de despachos. Por ejemplo que el primero de alguna ruta sea un determinado cliente.
- Existen múltiples bodegas, y los vehículos no necesariamente vuelven a la misma bodega de la que salieron.
- El despacho a algunos clientes puede ser optativo, penalizando la función objetivo.
- La función objetivo puede ser compleja, incluyendo términos dependientes de la distancia viajada, el numero de vehículos usados, la duración en tiempo de las rutas y penalidades por no entregar a algunos clientes, o por violar las ventanas de tiempo.
- Más de un producto a despachar, por lo cual las bodegas y los vehículos deben contener de todos los productos.
- Restricción a compartir 2 tipos de productos dentro de un viaje. Por ejemplo: Carga animal.

3.2.4 Ruteo de Vehículos con Ventanas de Tiempo (VRPTW)

El Problema de Ruteo de Vehículos con Ventanas de tiempo (VRPTW) es un VRP que tiene una restricción temporal. Dicha restricción es que los clientes deben ser servidos por los vehículos dentro de un rango de tiempo determinado, conocido como ventana de tiempo [28] [30].

Describiremos la ventana de tiempo de un cliente i por:

$[e_i,l_i]$: Intervalo de tiempo en el que se puede iniciar el servicio que entrega el vehículo

A_i : Tiempo de llegada del vehículo al cliente i

D_i : Tiempo de abandono del cliente i

S_i: Tiempo que toma el servicio al cliente i

T_{ij}: Tiempo de Viaje del cliente i al cliente j

Ya que el servicio debe tener lugar en la ventana de tiempo, se requiere que:

$e_i <= D_i <= l_i$ para todo i.

Si $a_i < e_i$ entonces hay un tiempo de espera de $w_i = e_i - a_i$, antes de que se inicie el servicio, hasta que se abre la ventana de tiempo.

El objetivo del VRPTW además de minimizar la flota y los tiempos/costos de viaje también es minimizar los tiempos de espera, recién definidos.

Las ventanas de tiempo pueden ser hard o soft. En el caso de las ventanas hard, si un vehículo llega antes del inicio de la ventana a un cliente, está permitido esperar hasta que el cliente este listo para recibir el servicio. Sin embargo, no está permitido llegar después del fin de la ventana al cliente. En contraste, en las ventanas de tiempo soft, las ventanas pueden ser violadas a un costo.

En algunos casos, es posible que un cliente tenga más de una ventana de tiempo, por ejemplo en el caso de despacho de insumos a un restaurant, se tienen dos ventanas de tiempo, antes de la hora de almuerzo, y después de la hora de almuerzo, ya que no se puede estar recibiendo mercadería en la hora peak, que es la hora de almuerzo.

Si la diferencia entre e_i y l_i es pequeña comparado con el horizontes de tiempo se habla de una ventana angosta. Hoy en día, para entregar un mejor servicio, cada vez más, se le ofrece al cliente ventanas de tiempo más angostas, ya que significan un ahorro de tiempo para los clientes, que no tienen que estar esperando a lo largo de grandes espacios de tiempo.

3.2.5 Otros Problemas VRP

VRP con Múltiples Bodegas (MDVRP)

En esta variación del problema de ruteo de vehículos, no se tiene una única bodega, sino muchas, donde cada una tiene asociada una flota de vehículos, que deben servir a los clientes [32]; por lo cual se tiene un problema adicional, que es asignar los clientes a las distintas bodegas. También se puede complejizar más el problema, permitiendo que los vehículos no necesariamente vuelvan a la misma bodega de la que partieron.

VRP con backhauls

El problema de ruteo con bakhauls es un VRP en el cual los clientes pueden recibir o entregar productos a los vehículos [16] [20]. Linehaul son los clientes que reciben productos de la bodega, y backhaul puntos son los clientes que envían productos de vuelta a la bodega. La asumsión critica es que todos los despachos deben ser hechos en cada ruta antes que cualquier devolución sea hecha. Esto viene del hecho que no es factible reordenar la carga de los vehículos durante la ruta.

Split Delivery VRP (SDVRP)

SDVRP es una relajación del VRP donde está permitido que el mismo cliente pueda ser servido por diferentes vehículos, si esto reduce el costo total [32].

Esta relajación es muy importante si los tamaños de las órdenes de los clientes son muy grandes en comparación con la capacidad de los vehículos

VRP con Despacho y Recolección (VRPPD)

El problema de ruteo de vehículos con Despacho y recolección es un VRP en el cual existe la posibilidad que los clientes devuelvan productos [32]. Es usual considerar algunas restricciones al problema, como que todos los productos que se despachan vienen de la bodega, y todos los productos que se devuelven van a dar a la bodega, o sea no se permite el intercambio de productos entre los clientes. Otra alternativa es relajar la restricción que todos los clientes sean visitados exactamente una vez.

En esta tesis el dominio de estudio será con una bodega, varias ventanas, demanda en los nodos, sólo un vehículo atiende a un cliente, y el resto de las características libres.

3.2.6 Técnicas de Solución del VRP

Como se ha visto el problema de ruteo de vehículos es de una alta complejidad matemática, lo que sumado a la gran cantidad de variantes de él que existen, nos presenta un escenario bastante complicado para poder obtener una solución adecuada.

Las técnicas de solución se dividen en [17]:

- Procedimientos Exactos
- Heurísticas
- Meta Heurísticas

3.2.6.1 Métodos Exactos

Dada la complejidad de los problemas, sólo las instancias con pocos clientes pueden ser resueltas consistentemente por métodos exactos. En este tipo de metodologías, suele resolverse alguna relajación del problema (ver problema en Anexo 1) usando por ejemplo Branch and Bound. También se han propuesto algoritmos basados en Programación Dinámica que aceleran los cálculos mediante una relajación del espacio de estados. Por otro lado algunos problemas, se pueden resolver usando métodos de generación de columnas.

3.2.6.2 Heurísticas

Las heurísticas, son procedimientos simples que realizan una exploración limitada del espacio de búsqueda y dan soluciones de calidad aceptable (no necesariamente óptimas) en tiempos de cálculo moderados. Son la forma más usada para resolver problemas de ruteo de vehículos, ya que como se sabe la mayorías de estos son NP-completos [32], lo que deriva en grandes tiempos de resolución cuando se usan métodos exactos.

Las heurísticas tienen también otra ventaja sobre los métodos exactos, que es que son menos sofisticadas algorítmicamente, con lo que es mas fácil programarla y a la vez son mas fáciles de comprender.

La mayoría de las heurísticas que se estudiarán [23], resuelven los tipos más sencillos de VRP, pero en los últimos años han sido extendidas/modificadas para manejar una gran cantidad de restricciones adicionales, como son ventanas de tiempo, largo máximo de la ruta, etc.

Existen cinco familias de heurísticas para resolver el VRP.

A. Ahorro

Se construye una solución de forma que en cada paso del procedimiento una configuración actual (que posiblemente es infactible) se compara con una

configuración alternativa que también puede ser infactible. Esta configuración alternativa es tal que entrega el más grande ahorro en términos de alguna función (por ejemplo: costo total). El proceso iterativo termina cuando se encuentra una configuración factible.

El más conocido de los método de ahorro es el de Clark & Wright (1964), el cual hasta hoy sigue siendo uno de los más usados.

B. Inserción

Son métodos en los cuales se crea una solución mediante sucesivas inserciones de clientes en las rutas. En cada iteración se tiene una solución parcial cuyas rutas sólo visitan un subconjunto de los clientes y se selecciona un cliente no visitado para insertar en dicha solución.

C. Mejora/Intercambio

Se mantiene en toda iteración la factibilidad y se busca la optimalidad. En cada paso una solución factible es modificada para avanzar hacia otra solución factible pero con un costo total menor. El procedimiento continua hasta que no se pueden hacer reducciones de costo.

D. Cluster First, Route Second

Los métodos asignar primero y rutear después (Cluster First, Route Second) tienen dos fases. En la primera se busca generar grupos de clientes, también llamados clusters, que estarán en una misma ruta en la solución final. Luego para cada cluster se crea una ruta que visite a todos sus clientes. Las restricciones de capacidad son consideradas en la primera etapa, asegurando que la demanda total de cada cluster no supere la capacidad del vehículo. Por lo tanto construir las rutas de cada cluster, es resolver un TSP, que dependiendo del número de clientes del cluster, puede resolverse de manera exacta o aproximada.

Uno de los métodos más usados para construir los clusters es el algoritmo Sweep o de Barrido.

E. Route First, Cluster Second

En los métodos rutear primero, asignar después (Route First, Cluster Second) también hay dos fases. Primero se calcula una ruta que visita a todos los clientes resolviendo un TSP. En general esta ruta no respeta las restricciones del problema, por lo cual en la segunda fase la gran ruta se particiona en varias rutas, cada una de las cuales si es factible.

3.2.6.3 Metaheurísticas

Para obtener mejores soluciones que las heurísticas presentadas en 3.2.6.2, es necesario recurrir a técnicas que realicen una mejor exploración del espacio de soluciones. Las metaheurísticas son procedimientos genéricos de exploración del espacio de soluciones para problemas de optimización y búsqueda. Proporcionan una línea de diseño que, adaptada en cada contexto, permite generar métodos de solución. En general, las metaheurísticas obtienen mejores resultados que las heurísticas clásicas, pero incurriendo en mayores tiempos de ejecución (que de todos modos, son inferiores a los de los métodos exactos).

Los métodos metaheurísticos más usados [22] son los algoritmos de hormigas, los algoritmos de búsqueda tabú y los algoritmos genéticos.

4. Descripción de un Sistema de Ruteo de Vehículos

4.1 El Sistema de Ruteo de Vehículos en el Ámbito Organizacional

En la Gestión de Operaciones una de las áreas importantes de estudio es la Logística, la cual frecuentemente es definida como el conjunto de funciones que permiten a las empresas llegar con sus productos a los clientes. En la literatura [11] se divide a la logística en tres subsistemas:

Logística de Procuración: Que tiene que ver con las actividades que las empresas deben realizar para abastecerse de los insumos para producir.

Logística Interna: Que tiene que ver como se producen al interior de la empresa los productos.

Logística de Distribución: Llegar con los productos a los clientes

Por otro lado existe el concepto de ciclo de vida de las órdenes [11], que se refiere a todos los pasos necesarios desde que un cliente solicita un producto/servicio hasta que este le es entregado/servido. Las etapas del Ciclo son:

Proceso de Venta:

- Elaboración de la Orden por parte del cliente
- Transmisión de la Orden

- Recepción de la Orden
- Aceptación/Rechazo de la Orden
- Producción del Producto y Documentos de Despacho
- Generación y envío de Factura

Inventario o Almacenaje:

- Control y Planificación del Inventario
- Planificación del Almacenaje
- Carga/Descarga de Vehículos

Transporte:

- **Ruteo y Scheduling de Vehículos (SRV)**
- Flota de Vehículos
- Pago a los choferes

También se hace una distinción del tipo de decisiones que se deben tomar, en cuanto a la dimensión temporal. Es así, como existen: decisiones estratégicas, decisiones tácticas y decisiones operacionales. El Ruteo de Vehículos como toda función de logística, también está enmarcada en estos tres tipos de decisiones, y está compuesta de diversas tareas en cada uno de ellos [11]:

Estratégico (Planificación 1-3 años)

- Planificar número, tamaño y ubicación de bodegas
- Determinar el nivel de servicio al cliente

- Planificar Tamaño y Composición de la Flota de Vehículos

Táctico (Planificación menos de 1 año, mas de 1 semana)

- Asignación de clientes a centros de distribución
- Planificación de sectores de distribución y grupos de clientes
- Designación de días y horas de despacho
- Decisión de usar vehículos propios o contratados

Operacional (menos de 1 semana)

- Generación de Rutas Diarias/Semanales
- Asignación diaria/semanal de ordenes a rutas fijas
- Optimización de rutas
- Inserción de ordenes de último minuto
- Cálculo de Costos de Transporte

En este trabajo principalmente **se estudiará el nivel operacional del Ruteo de Vehículos**.

4.2 Funcionalidades de un Sistema de Ruteo de Vehículos Computarizado (SRVC)

Un sistema de ruteo de vehículos debe contar con ciertas funcionalidades, que permitan cumplir con su objetivo de ayudar a los usuarios en la toma de decisiones respecto a como servir a los clientes con la flota que se dispone [2] [17] [19].

Los sistemas de ruteo de vehículos en un principio eran manuales, donde el tomador de decisiones en base a su experiencia práctica, debía decidir qué vehículo servía a cada cliente, ayudado solamente por un “mapa en la pared” y cálculos matemáticos en papel, para así construir las rutas y estimar los costos. Como es de esperar, este mecanismo en la mayoría de las veces está lejos de generar soluciones óptimas, provocando gastos innecesarios para las organizaciones. Hoy en muchas de las pequeñas y medianas empresas sigue haciéndose así, por lo cual surgen los Sistemas de Ruteo de Vehículos Computarizado (SRVC)

4.2.1 Funcionalidades de un SRVC

Al revisar software existente y un estudio [19] que compara las decenas de software que hay en el mercado se puede llegar a que las principales funciones de un SRVC son:

i) Funciones Relacionadas con Input/Output

Ingreso de Datos

Los datos pueden ser ingresados interactivamente en la interfaz gráfica o a través de archivos de datos con cierta estructura predefinida:

Vehículos: cantidad, capacidad (volumen y/o peso), tipo del vehículo, otras restricciones

Bodega: ubicación, capacidad, otras restricciones

Clientes: ubicación, demanda, otras restricciones

Visualización

Prácticamente todos los actuales software de ruteo de vehículos incorporan un módulo para desplegar en pantalla el mapa de la zona de estudio, la ubicación de la bodega, la ubicación de los clientes, y las rutas que propone el sistema. Este despliegue permite entre otras cosas: borrar o agregar clientes, visualizar zonas urbanas (calles), examinar las rutas y ajustarlas.

Cálculo de Tiempos/Distancias

Uno de los cálculos fundamentales que debe hacer un SRVC es la distancia entre dos puntos del área de estudio. Para eso la alternativa obvia es calcular la distancia euclidiana entre los 2 puntos. Cuando lo que se desea optimizar es el tiempo, se debe calcular en base a la distancia que hay entre 2 puntos. Para eso el usuario debe indicar con que velocidad se mueven los vehículos. Si se quiere complejizar más el calculo, la velocidad puede ser distinta en diferentes lugares, por ejemplo en zonas urbanas la velocidad es mucho menor que en carreteras.

Bases de Datos de Calles/Direcciones

La ubicación de la bodega y los clientes generalmente son coordenadas (x,y), pero también se debe dar la posibilidad que sean direcciones reales en una ciudad, las cuales son obtenidas de sistemas de información geográficas que tienen bases de datos con las calles. También una alternativa que es frecuentemente usada es el código postal (zip code). Los SRVC deben conectarse a estos sistemas y convertirlos en coordenadas en un mapa.

Detalle Nivel Calle

Cuando se cuenta con la información recién descrita, se puede hacer un ruteo a través de las calles, tomando en cuenta información de restricciones al tránsito como: calles de una sola dirección, no entrar, no doblar, dirección obligada.

Reportes

Una vez que se genera una solución (conjunto de rutas) para el problema, se pueden entregar diversos reportes al usuario, que le permiten evaluar lo propuesto por el software: distancia total recorrida por la flota al día/mes, distancia/tiempo promedio de cada ruta, tasa de ocupación de los vehículos, costo en pesos de la solución, etc.

ii) Funciones Relacionadas con Restricciones del VRP

Ventanas

Como vimos previamente las ventanas de tiempo son los intervalos de tiempo en el que el cliente desea recibir el servicio. Estas pueden ser hard (el cliente debe ser visitado en el intervalo) o soft (hay cierta holgura respecto al intervalo). También esta siendo más común el uso de más de una ventana por cliente. En la actualidad, las ventanas de tiempo cada vez son más estrechas, ya que significan un mejor servicio al cliente.

Planificación Multiperíodo

El software de ruteo de vehículos no necesariamente se está corriendo en el día a día, sino que puede tener otro horizonte de tiempo, por ejemplo una semana, donde si un cliente no es visitado un día, podrá serlo al día siguiente, y se

considera a la noche como un período en el que no se pueden hacer despachos, y los vehículos pueden o no volver a las bodegas.

Múltiples Bodegas y Flotas Integradas

Existen problemas más complejos en los que hay más de una bodega, y el software debe permitir optimizar el ruteo como un todo, asignando clientes a bodegas y permitiendo en muchos casos que los vehículos no retornen a la bodega de origen.

Manejo de la Tripulación

Los software más modernos también están incorporando elementos que permiten manejar en conjunto, la tripulación de la flota de vehículos, con todo lo que ello significa: hora de almuerzo, no se puede manejar mas de un cierto numero de horas ininterrumpidamente, costo de horas extras, etc.

5. Análisis del Framework

En este estudio se aplicará el paradigma Orientado a Objetos para el análisis, diseño y construcción de un framework -no de un software en específico- por lo cual la metodología deberá sufrir ciertas modificaciones, como por ejemplo agregar el diseño de los hotspots y un análisis de requerimientos más genérico. Por otro lado el enfoque de la construcción será top-down, o sea, partiendo desde abstracciones de un problema genérico de ruteo de vehículos y bajando hacia elementos más concretos del VRP.

5.1 Análisis de Requisitos

El análisis de requisitos busca establecer qué desea el usuario. Al utilizar el enfoque Orientado a Objetos, los requerimientos se ven no como una lista secuencial de cosas que el usuario desea, sino quién utilizará el sistema y qué actividades realizará frente a los requerimientos del usuario. Para especificar o caracterizar la funcionalidad y comportamiento de la aplicación se utiliza el diagrama de Casos de Usos de UML [21].

El modelo de casos de usos es un modelo donde intervienen Actores y Casos de Uso. Un actor representa a un humano, máquina u otro sistema. El caso de

uso corresponde a un diálogo entre actores y el sistema, que siempre es iniciado por los actores quienes invocan funcionalidades específicas.

La identificación de los casos de uso, se hizo basándose en el estudio del dominio del problema del capítulo 3, donde se identificaron distintos tipos de problemas, la características fundamentales, los actores, etc.; y en el capítulo 4 donde se estudiaron sistemas/software existentes de ruteo de vehículos.

Identificación de Actores

Solamente hay un actor, que es el encargado del despacho de camiones, quien puede ser un simple usuario al que se le capacita en el uso del software o un jefe/gerente de despacho, que conoce bien cómo funciona el despacho de vehículos.

Casos de Usos

Se identifican tres funcionalidades centrales:

- Ingreso de datos del VRP a resolver (clientes, vehículos, etc)
- Solución del VRP
- Modificar los datos ingresados (generalmente después de la resolución del problema)

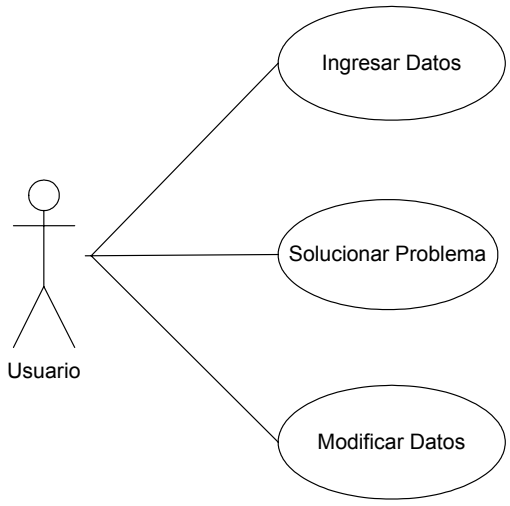


Figura 4: Casos de Usos

INGRESO DE DATOS

Ingresar Clientes	Se ingresan todos los datos de cada uno de los clientes
Ingresar Clientes – Posición	El usuario especifica la posición de un cliente, generalmente un par ordenado (x,y), pero que también podría ser una dirección o un zipcode.
Ingresar Clientes – Ventana	Los clientes pueden/deben ser visitados en cierto rango de tiempo. Se debe especificar cual es el intervalo (ventana)
Ingresar Clientes – Demanda	Los clientes requieren que les lleven/retiren productos. La demanda podría ser de más de un producto.
Ingresar Clientes – Duración Servicio	En algunas ocasiones se puede estimar la duración del servicio.

Ingresar Bodega	Se ingresan los datos que caracterizan a la bodega.
Ingresar Bodega – Posición	Al igual que los clientes, su ubicación generalmente esta dada por un par (x,y)
Ingresar Bodega – Capacidad	En algunos casos, es importante conocer la capacidad de la bodega, sobre todo cuando el problema es de retiro de productos de los clientes.

Ingresar Vehículos	Se ingresan las características de los vehículos.
Ingresar Vehículo – Capacidad	Capacidad de transporte del vehículo.
Ingresar Vehículo – Máximo Ruteo	El vehículo no puede viajar mas de una distancia/tiempo máximo.

Ingresar Parámetros	Además de los elementos centrales como lo son la bodega, clientes y vehículos, los problemas de ruteo de vehículos tienen otros parámetros que los caracterizan.
Ingresar Parámetros – Máximo Ruteo	Las rutas de la solución no pueden superar un máximo de distancia/tiempo.
Ingresar Parámetros – Velocidad	Cuando corresponda, se debe especificar la velocidad con que viajan los vehículos.
Ingresar Parámetros – Tipo Ventanas	Las ventanas pueden ser soft o hard.
Ingresar Parámetros – Función Objetivo	En algunos casos, la función objetivo es una fórmula dependiente de la distancia, tiempo, e incluso del número de vehículos usados.

SOLUCIÓN DEL VRP

Calcular Matriz de Costos	Se debe calcular la distancia o costo de viajar de un cliente a otro. Puede ser que no haya que calcular todas las distancias entre clientes, sino sólo algunas.

Corre Algoritmo	El usuario una vez ingresados todos los datos, pide al sistema que le resuelva el problema.

Guardar Solución	Es de mucha utilidad almacenar la solución que entregó el sistema, por ejemplo, para poder compararla con soluciones de problemas similares, permitiendo que el usuario modifique ciertos datos.

Entregar Solución	El output de la resolución del problema es un conjunto de rutas, las cuales pueden ser entregadas numéricamente o gráficamente en un mapa de la zona geográfica de estudio.

Figura 5: Caso de Uso: Ingreso de Datos

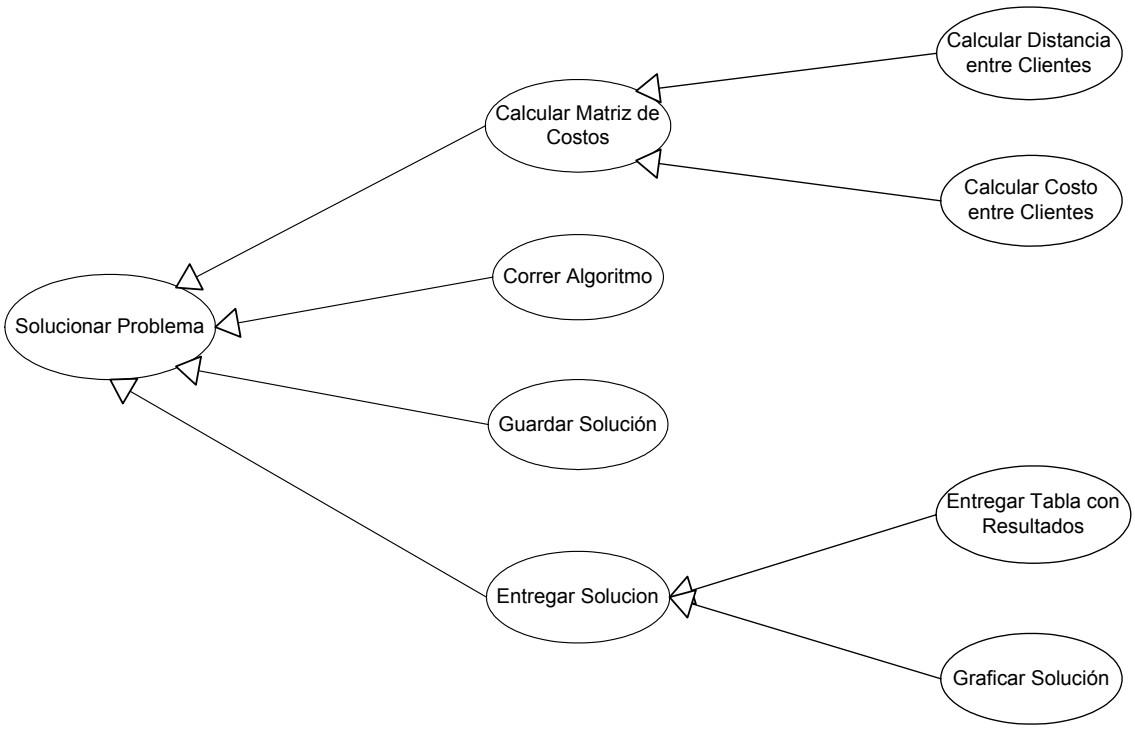


Figura 6: Caso de Uso: Solucionar Problema

MODIFICACION DE DATOS

Son los mismos casos de uso del ingreso de datos, a excepción de:

Agregar Cliente	Se pueden agregar clientes a los ya ingresados inicialmente.
Eliminar Cliente	Vistos los resultados, el usuario podría decidir eliminar algunos clientes.

Agregar Vehículo	Se pueden agregar vehículos a los ya ingresados inicialmente.
Eliminar Vehículo	Vistos los resultados, el usuario podría decidir eliminar vehículos.

5.2 Modelo Conceptual

Basándose principalmente en el análisis de requisitos, se obtienen las clases que se necesita para modelar el framework. El modelo conceptual (Figura 7) muestra gráficamente, a través de un grupo de elementos, los conceptos, los atributos y las asociaciones más importantes del dominio.

PROBLEMA: Es la clase principal del framework.

FLOTA: Conjunto de vehículos

cantidad: número de vehículos disponibles

BODEGA: Desde donde parten/llegan los vehículos

capacidad: Capacidad de la bodega

posicion: Ubicación de la bodega

VEHÍCULO: Recorre los clientes entregando productos

id: identificador del vehículo

capacidad: capacidad del vehículo

CLIENTE:

id: Identificador del cliente

posicion: Posición del cliente

demanda: Demanda del cliente

TOUR: Un vehículo recorre un tourcantidad: Número de clientes del tour

carga: Cantidad de productos que transporta el vehículo

cantidad: Número de clientes del tour

distancia: Distancia que cubre el tour

PLANRUTEO: Conjunto de toures que componen la solución

cantidad: Número de rutas

distancia: Distancia que cubre el plan de ruteo

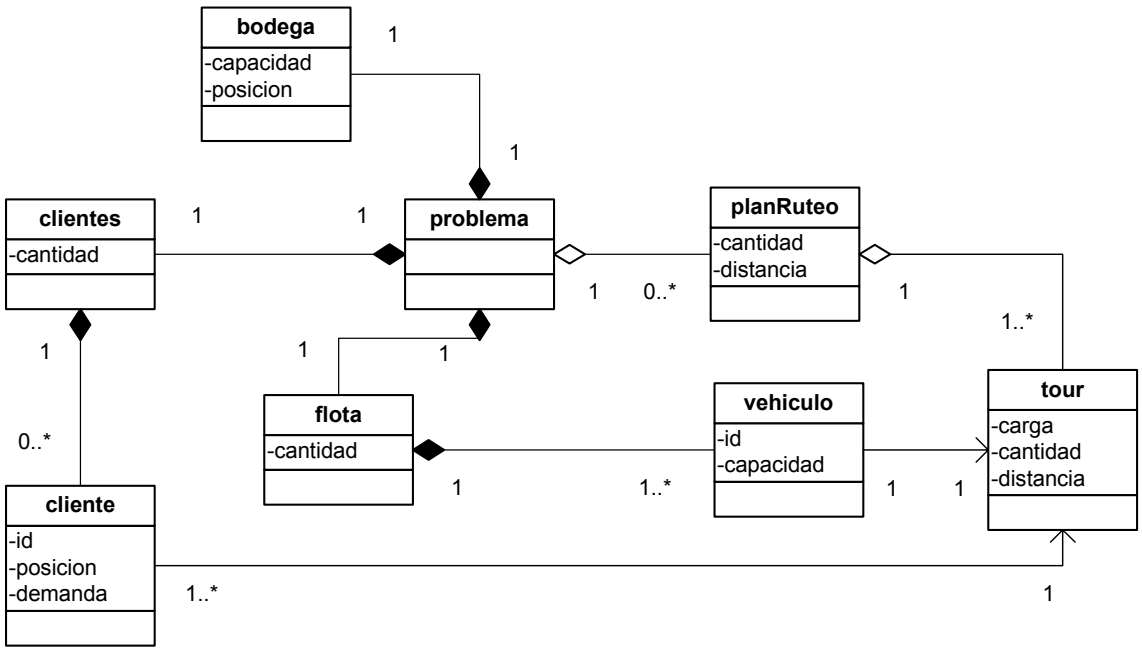


Figura 7: Modelo Conceptual

6. Diseño del Framework

6.1 Diagrama de Actividades del Framework

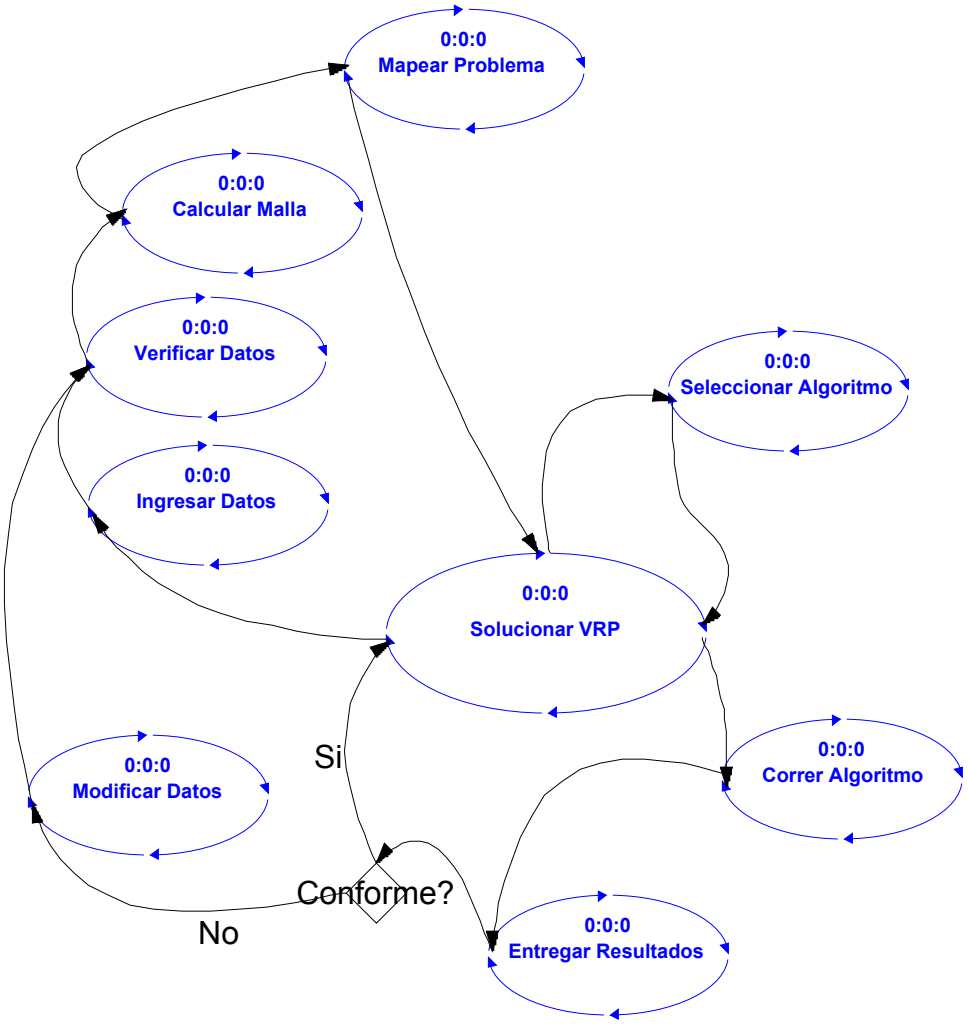


Figura 8: Diagrama de Actividades del Framework

Ahora que ya se analizaron los requerimientos a satisfacer, se debe definir el funcionamiento de la solución. O sea, cuales son los pasos para resolver un problema de ruteo de vehículos, los cuales se pueden ver en la Figura 8.

Ingresar datos

De los clientes, vehículos y bodega

Verificar Datos

Se corroborará que los datos ingresados estén de acuerdo al tipo de problema y sus características. Por ejemplo si se especificó que se transportarían varios productos, la demanda de los clientes y la capacidad de los vehículos, también debe constar de varios productos.

Calcular Malla

La malla es el input principal de los algoritmos de VRP, la cual contiene la distancia entre todos los clientes. Dadas las posiciones de los clientes se deben calcular la distancia entre ellos

Mapear Problema

Una vez ya conocido el problema y sus características se debe hacer un match con el conjunto de algoritmos con que se cuenta (Ver Cap. 7), o sea si el

problema a solucionar es del tipo A, se buscan algoritmos que resuelven los problemas del tipo A.

Seleccionar Algoritmo

En caso de haber más de un algoritmo que resuelve el problema, se le da a elegir al usuario.

Correr Algoritmo

Se aplica el algoritmo seleccionado, para obtener la solución.

Entregar Resultados

Se deben entregar al usuario los toures que componen la solución. Se deben entregar numéricamente y gráficamente en el mapa del área.

Modificar Datos

Puede ser que el usuario no este conforme con los resultados, por lo que deseará modificar alguno de los datos del problema.

6.2 Diseño de los Hotspots

Como se estudió en el capítulo 3, lo que caracteriza a los frameworks es la capacidad de generar diversas aplicaciones para un dominio de estudio. Esto se puede hacer a través de los hotspots [13], que son las partes variables del framework. Los hotspots son las clases y/o métodos abstractos que deben ser implementados en las aplicaciones que derivan del framework.

En nuestro problema existen varios hotspots, siendo el más importante el método de resolver el problema. Este hotspot se estudiará con detalle en el capítulo 7.

6.2.1 Calcular Malla

La malla almacena los costos/distancias de viajar de un cliente a otro. En la mayoría de los métodos es necesario que estén todas las combinaciones (n^2 valores), pero también hay muchos casos en que para cada cliente se requieren sólo algunas distancias, por ejemplo los p clientes más cercanos.

En la mayoría de los casos la malla debe ser calculada por el sistema, pero también puede ser que sea ingresada directamente. Para el cálculo de la

distancia entre dos clientes generalmente se ocupa una expresión dependiente de la distancia euclidiana, ya que los caminos no van en línea recta de un cliente a otro. Por ejemplo:

$$d = \alpha * (dist.euclidiana)^{\beta}$$

En los problemas de ruteo, el costo que nos interesa minimizar puede ser la distancia o el tiempo de viaje. Para calcular este último es necesario que sepamos cuál es la velocidad de viaje de los vehículos, la cual podría ser dependiente del vehículo o de la zona en que se viaja (urbana o carretera).

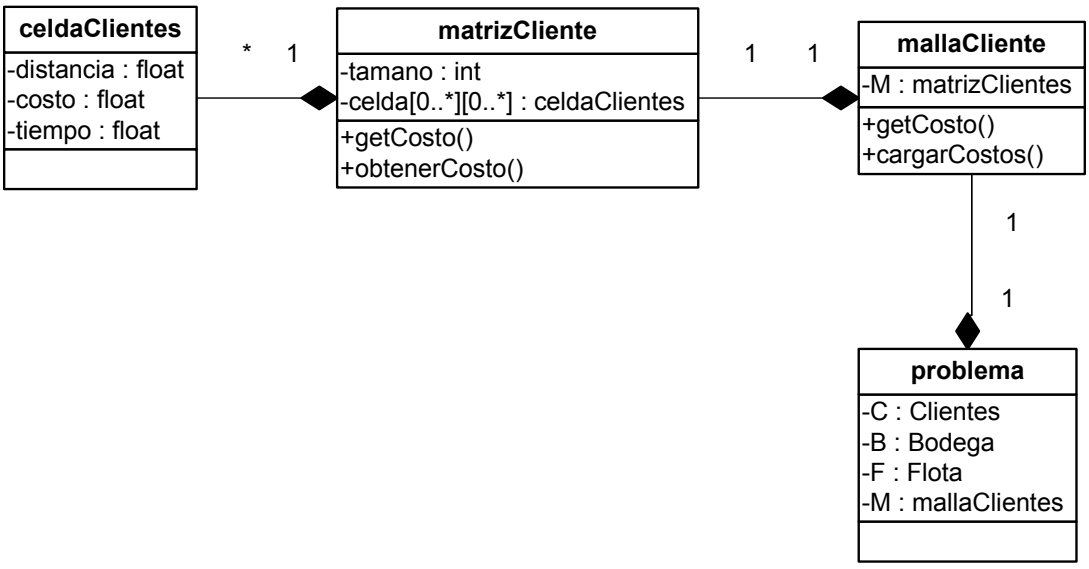


Figura 9: Hotspot 'Malla'

Entonces, **heredando** de las clases MatrizCliente y CeldaCliente (Figura 9) podríamos tener otra forma de calcular la malla del problema.

6.2.2 Capacidad

En los problemas de ruteo, se deben satisfacer las demandas por productos de los clientes. Los productos tienen ciertas características que nos permiten saber cuándo es posible el transporte, cómo son su peso y volumen.

Los requerimientos de los clientes pueden ser de uno o más productos, siendo lo más usual que sólo sea uno.

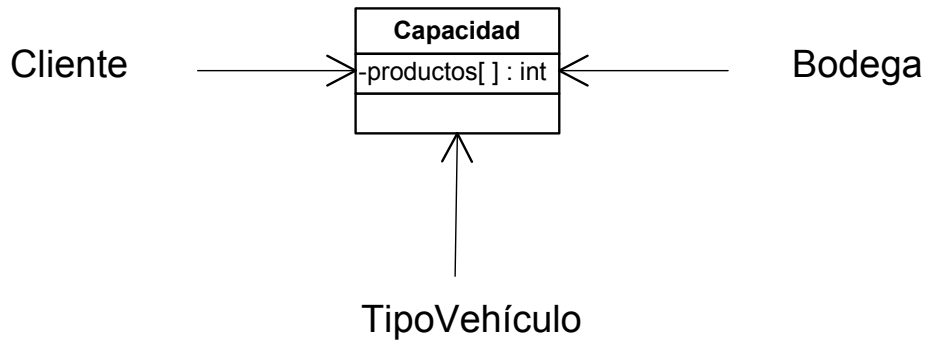


Figura 10: Hotspot 'Capacidad'

6.2.3 Tipos de vehículos

Es bastante común que las flotas de vehículos sean heterogéneas, pero por lo general hay un par de tipos de vehículos. Es importante identificar las características de los tipos de vehículos ya que puede ser que ciertos tipos pueden servir para ciertos clientes, por ejemplo para clientes de zonas urbanas no se usarían grandes camiones, más bien vehículos pequeños.

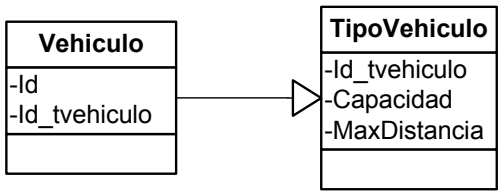


Figura 11: Hotspot 'Tipos de Vehículos'

6.2.4 Posición

La posición de los clientes y la bodega, generalmente se expresa como un par ordenado (x , y) en algún sistema de referencia externo (por ejemplo paralelos y meridianos). También debe tenerse un sistema de referencia interno del software, acotado al área de estudio, donde algunas veces la bodega es el centro.

Por otro lado puede darse el caso que la ubicación no esté expresada en coordenadas numéricas, por ejemplo, nombres de calles o códigos zip. En estos casos se debe transformar esta posición a puntos (x , y). Solo es cosa de modificar la clase Posición y sus métodos.

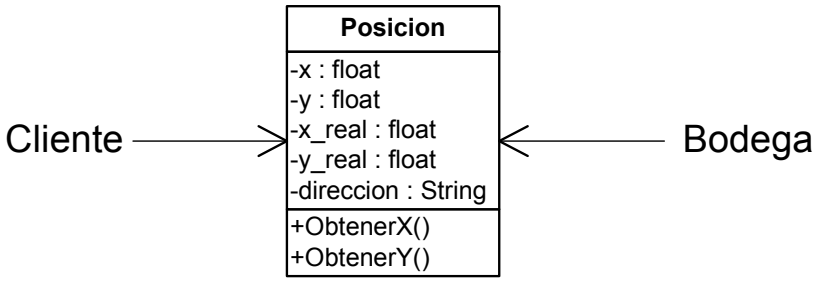


Figura 12: Hotspot 'Posición'

6.3 Hotspot Principal

Como se estudió en el capítulo 3 hay muchos tipos de problemas de ruteo de vehículos: con flota heterogénea, con ventanas, con ruteo máximo, en función del tiempo, con múltiples bodegas, etc., y por otro lado existen algoritmos específicos para cada uno de ellos, por lo cual se hace indispensable –como se viera en 6.1 "diagrama de actividades" – mapear los problemas con los algoritmos.

Para eso, se ideó un sistema en que tanto problemas como algoritmos tienen una serie de características, las cuales son del tipo:

CARACTERISTICA=VALOR

Ejemplos de características son:

FLOTA: Indica si es homogénea o heterogénea

VELOCIDAD: Indica que los viajes se realizan con la velocidad que se indica

VENTANAS: Indica cuantas ventanas tiene cada cliente

TIPO_VENTANA: Hard o Soft

DEMANDA: Indica cuantos tipos de productos constituyen la demanda

BAKCHAUL: Indica que el tipo de problema es con backhauls

MAXIMA_TIEMPO_RUTEO: Máximo que puede demorar el viaje de un vehículo

DURACIÓN: Se indicara cuanto demora el servicio de cada cliente

Por otro lado, se determinó que cada problema a resolver está en un archivo de texto que tiene tres partes, primero están las 'caracteristicas' del problema, luego la lista de clientes a visitar y finalmente los vehículos.

Veamos un ejemplo de archivo:

```
%PROBLEMA
DEMANDA=SI/1/1,1
VENTANAS=2//1,0
FLOTA=HETEROGENEA//0,0
DURACION=SI//1,0
VELOCIDAD=SI/50/0,0
CANTIDAD_VEH_TIPO=SI//0,1
%CLIENTES
2,6;VENTANAS =1-2,4-6;DEMANDA=12;DURACION=1
0,9;DEMANDA=35; VENTANAS=1-7,10-14;DURACION=3
9,5;VENTANAS=3-4,12-14;DEMANDA=7;DURACION=2
10,15;VENTANAS=4-5,6-7;DEMANDA=14;DURACION=2
2,0;VENTANAS=3-6,12-17;DEMANDA=34;DURACION=4
4,5;VENTANAS=1-3,9-12;DEMANDA=54;DURACION=6
%VEHICULOS
camiontolba;DEMANDA=100;CANTIDAD_VEH_TIPO=5
sportvagon;DEMANDA=30;CANTIDAD_VEH_TIPO=10
camionchico;CANTIDAD_VEH_TIPO=7;DEMANDA=50
```

Como se puede ver en la sección problema, las características además traen otros elementos. El segundo valor entre slashs es un subvalor que en ocasiones es necesario, como es el caso de la característica VELOCIDAD en el ejemplo.

El último valor de las características (después del ultimo flash) es el indicador de si esta afecta o no a los clientes y a los vehículos (separado por coma). Por ejemplo en el caso de las VENTANAS estas afectan sólo a los clientes.
Por el otro lado los algoritmos también tienen que decir que tipo de problemas resuelven, para lo cual se debe indicar las características y sus valores. Esto se hace a través de la clase carga Características:

```
algoritmo_ClarkWright a1=new algoritmo_ClarkWright();
a1.setNombreGlosa("Clark & Wright","Uno de los algoritmos clasicos");
a1.cargaCaracteristicas();
lista_algoritmos.addElement(a1);

public void cargaCaracteristicas() {
    creaCaracteristica("DEMANDA","1");
    creaCaracteristica("FLOTA","HOMOGENEA");
    creaCaracteristica("CANTIDAD_VEH_TIPO","SI");
    creaCaracteristica("DEMANDA","SI");
}
```

Ahora que problema y algoritmo tienen sus características definidas, se hace el mapeo, donde un algoritmo es candidato a resolver el problema si tiene exactamente el mismo conjunto de pares caracteristica=valor.

6.4 Diagrama de Clases

En el capítulo 5 se había visto un modelo conceptual del problema de ruteo de vehículos, a partir de él y a partir del diseño de los hotspots, se desarrolla un diagrama de clases (Figura 13), en el cual se detalla a nivel físico el framework. El diagrama de clases es el diagrama principal del diseño del sistema, el cual presenta las clases del sistema con sus relaciones estructurales y de herencia, detallando además los atributos y operaciones de cada objeto.

Ahora se detallarán aquí las principales clases del framework con sus atributos y métodos más importantes. Asimismo se indica (con un rombo) cuales métodos hay que escribir/sobreescribir en una aplicación que usa como base el framework.

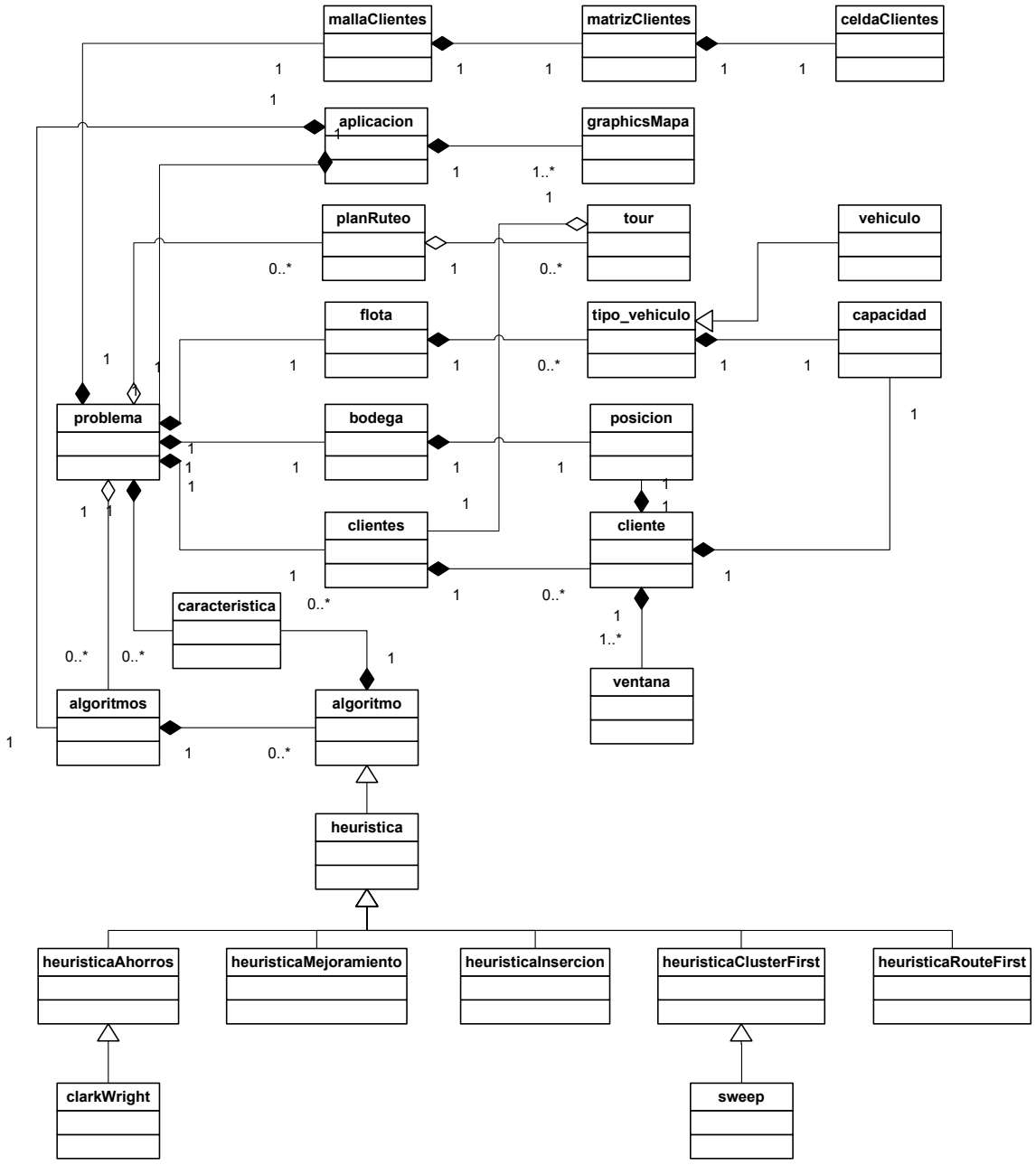


Figura 13: Diagrama de Clases

Problema

clientes c
bodega b
flota f
mallaClientes m
planRuteo pr // plan de ruteo que resuelve el problema
vector caracteristicas // lista de caracteristicas (nombre=valor) del problema

algoritmo

vector caracteristicas // lista de caracteristicas del algoritmo
String nombre
String glosa_descripción

cargaCaracteristicas()

Crea características para un algoritmo, llamando al método creaCaracteristica. Es un método que deben implementar los algoritmos finales, los que heredan de los 5 tipos de heurísticas. Si la aplicación que deriva del framework tiene algoritmos, se debe detallar obligatoriamente cuales son las características

creaCaracteristica(String nombre, String valor)

Para un algoritmo en especifico le crea una característica: nombre=valor

correr()

Es el cuerpo del algoritmo, obtiene un plan de ruteo para el problema.

algoritmos

vector lista_algoritmos

agregaAlgoritmos() 

Aquí se agregan todos los algoritmos con que se cuenta, indicando además nombre y una pequeña glosa.

Aplicación

problema p // Problema a resolver
algoritmos a // Algoritmos disponibles para correr
graphicsMapa gm

leeClienteArchivo(String linea, int cont)

Lee desde línea todas las características que tiene este cliente, para posteriormente crearlo (posición y características).

leeVehiculoArchivo(String linea, int cont)

Lee desde línea todas las características que tiene este vehículo, para posteriormente crearlo (nombre y características).

mostrarReporte(jInternalFrame jif)

Muestra en una textarea el reporte de la solución obtenida, o sea que cliente está en que tour, y cuanta carga contiene cada tour, entre otras cosas (asignación de vehículos, tiempo de visita de cada cliente, etc)

manejarCorrerMenu()

Es la función principal que se ejecuta cuando el cliente elije solucionar el problema. Entre otras operaciones limpia la solución, en caso de que existiese; busca los algoritmos que calzan con el tipo de problema; da a elegir al usuario que algoritmo usar para solucionar el problema; corre el algoritmo elegido; grafica la solución.

manejarCargarMenu()

Lee la información del problema que está en el archivo. Primero limpia todos los datos, por si hubieran quedado de un problema anterior. Posterior a eso, va leyendo línea por línea el archivo del problema (Ver 7.1). Primero lee los datos que caracterizan al problema, luego los datos de los clientes y finalmente los datos de los vehículos. Durante la lectura del archivo, va verificando que el problema tenga la sintaxis correcta y que además sea consistente, o sea por ejemplo si decimos que el problema tiene dos ventanas, los clientes efectivamente deben tener dos ventanas.

manejarDibujarMenu()

Dibuja en el mapa todos los clientes.

inicializar()

Inicializa los elementos gráficos de la aplicación (menus, etc)

bodega

capacidad cap

posición pos

caracteristica

int id

String nombre

String valor

String subvalor

int afecta_clientes // Si la característica debe estar en los clientes

int afecta_vehiculos // Si la característica debe estar en los vehículos

verificaConsistenciaGuardaCliente(int cont_id, String par_caract)

Esta función es llamada por leeClienteArchivo para verificar que las características que vienen con el cliente sean consistentes. A la vez se guardan cada una de las características del cliente.

verificaConsistenciaGuardaVehiculo(int cont_id, String par_caract)

Esta función es llamada por leeClienteVehiculo para verificar que las características que vienen con el vehículo sean consistentes. A la vez se guardan cada una de las características del vehículo.

cliente

int id

posición pos

capacidad demanda

generarReporte(int id_cli) 

Agrega al reporte de la solución la información del cliente: posición y demanda.

Si el cliente es la bodega (id_cli=0) no corresponde mostrar la demanda.

clientes

vector lista_clientes

int cantidad // número de clientes del problema

flota tamano_realidad // tamaño real del mapa

crearCliente(int ide, flota x, flota y, flota dda)

Se crea un cliente con id ide, posición (x,y) y demanda dda.

flota

vector lista_tipo_vehiculos

int cantidad_tipo_vehiculos

int cantidad_total_vehiculos

crearTipoVehiculo(int cont, String modelo)

Se crea un nuevo tipo de vehículos y se agrega a la lista de tipos de vehículos.

Es útil cuando la flota es heterogénea.

graphicsMapa

float tamano_mapa // Tamaño del mapa a desplegar en pixeles

dibujarClientes()
Dibuja en el Canvas los clientes del problema

dibujarRutas()
Dibuja en el Canvas cada una de las rutas de la solución.

mallaClientes

matrizClientes m

cargarCostos()
Calcula las distancias/tiempos/costos de ir a todos los clientes entre si.

calcularDistancia (int i, int j)
Calcula la distancia/tiempo/costo de ir del cliente i al j.

matrizClientes

int tamano // dimension de la matriz
celdaCliente celda[][]

calcularDistancia(int id_cl1, int id_cl2)

Calcula la distancia/tiempo/costo de ir del cliente id_cl1 al cliente id_cl2

obtenerDistancias()

Carga la matriz de celdaCliente's con la distancia/tiempo/costo entre los clientes.

planRuteo

vector toures // Toures que componen el plan de ruteo
int cantidad // cantidad de toures
String rpte

generarReporte()

Agrega al reporte de la solución la información del plan y llama a generarReporte de los toures.

agregaTour(tour t)

Agrega un tour al plan de ruteo

buscaTour(int cli_id)

Busca en el plan de ruteo en que tour está el cliente.

tipoVehiculo

int id
String modelo
capacidad cap

vector lista_vehiculos
int cantidad_vehículos

crearVehiculo(int id)
Se crea un nuevo vehículo y se agrega a la lista de vehículos del tipoVehiculo.

tour

int id_vehiculo // vehículo que lo cumple
vector clientes // Clientes que componen el tour
int cantidad // Número de clientes del tour
capacidad carga // Carga que tiene el tour

dibujaTour(int t_id, Graphics g)
Dibuja el tour t_id en el Canvas.

generarReporte()
Agrega al reporte de la solución la información del tour y llama a generarReporte de los clientes que lo componen.

agregarClientePosicion(cliente cl, int donde)

Agrega el cliente cl en la posición donde

6.5 Diagramas de Secuencia

Como se vió en el capítulo 5, los dos principales casos de uso, son el ingreso de datos y la solución del problema. En las Figuras 22 y 23 se muestran los **diagramas de secuencia** [21] de estos, los cuales modelan el aspecto dinámico de una aplicación, a diferencia de los modelos de clases y casos de uso que muestran sólo la parte estática.

Un diagrama de secuencia UML muestra las interacciones entre los objetos, organizados en una secuencia temporal. En particular muestra los objetos participantes en la interacción y la secuencia de mensajes intercambiados. En los diagramas de secuencia, la línea de vida de un objeto es la línea discontinua vertical, que representa la existencia de un objeto a lo largo de un período de tiempo. El foco de control es un rectángulo delgado que representa el período de tiempo durante el cual un objeto ejecuta una acción. Los mensajes se muestran como flechas entre líneas de vida.

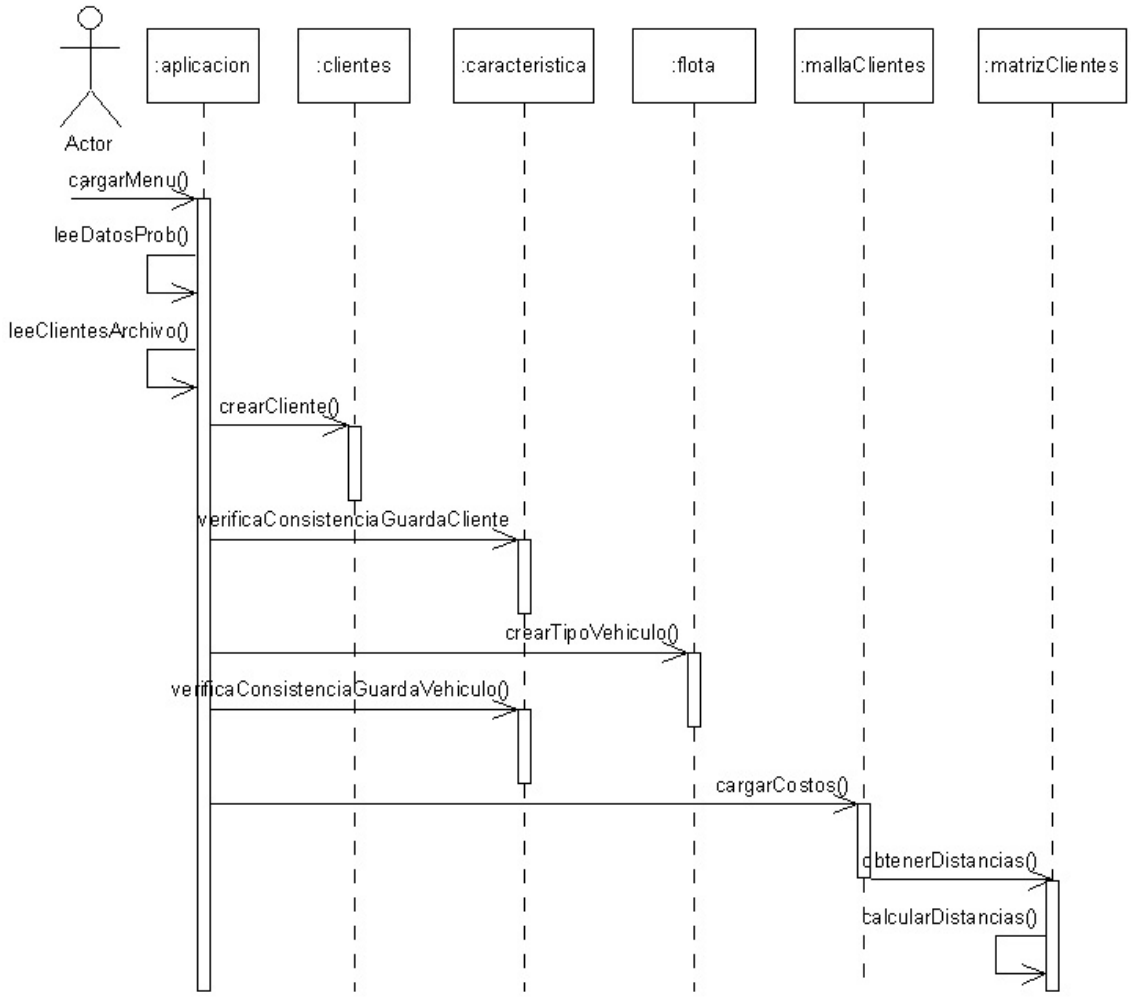


Figura 14: Diagrama de Secuencia de Ingresar Datos

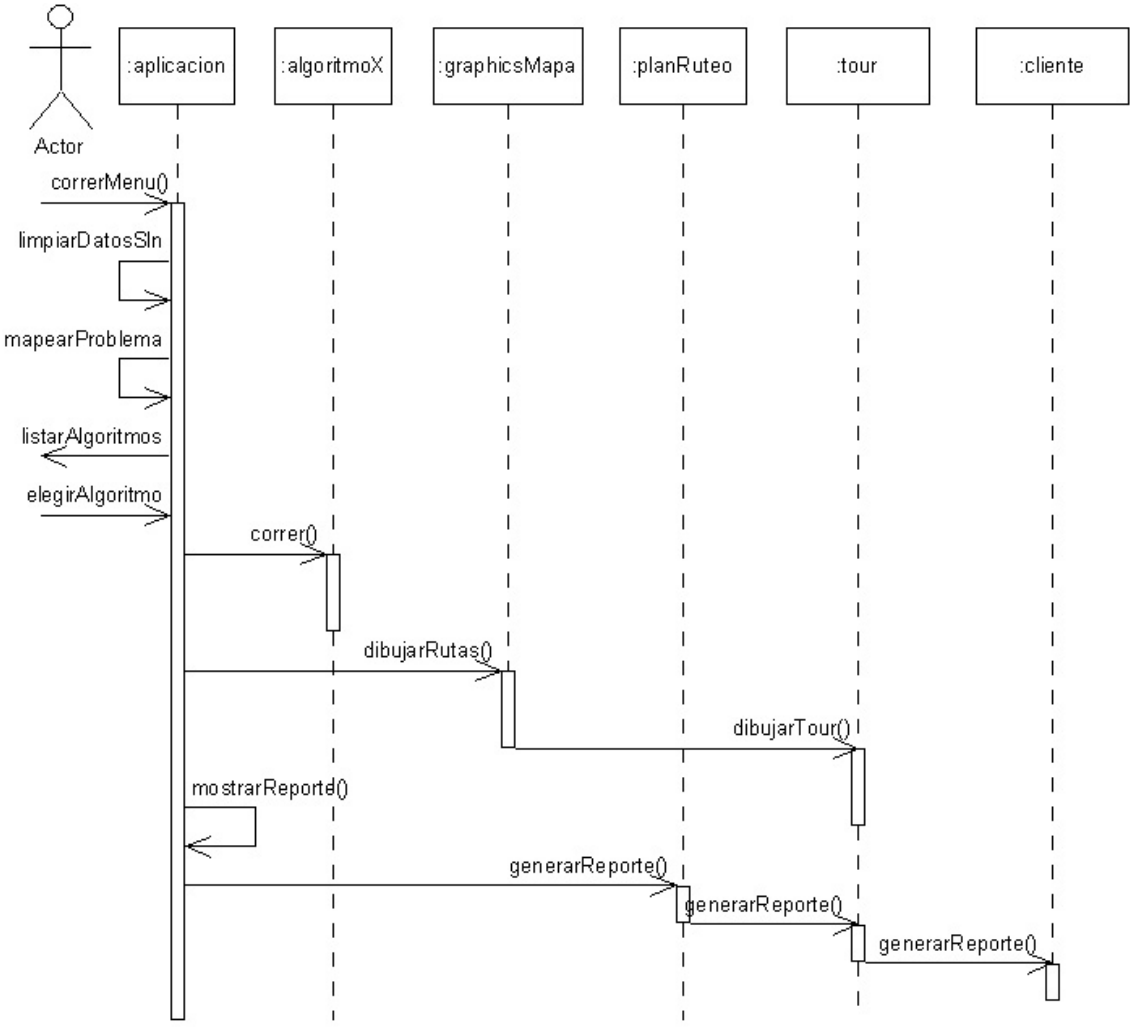


Figura 15: Diagrama de Secuencia de Resolver Problema

7. Métodos de Solución del VRP

Como se vió en el Capítulo 6, lo más importante de un sistema de ruteo de vehículos computarizado (SRVC) es el algoritmo con que se resolverá el problema. Una vez determinado el tipo de problema, se busca los algoritmos adecuados para resolverlo. Estos algoritmos reciben como input los datos de los clientes, de la bodega, y de los vehículos, y generan como output un conjunto de rutas que sirven a los clientes.

En los anteriores capítulos se desarrollaron elementos del framework como el modelo conceptual, el diagrama de actividades y diagrama de clases que se usa para resolver un VRP. También se estudiaron algunos hotspots como son el formato de ingreso de los datos y el cálculo de las distancias/costos.

En este capítulo, se estudiará el más importante de los hotspots – o puntos de variabilidad – del framework, que es la forma matemática en que se resuelve el problema. O sea, debemos buscar la estructura genérica que tiene un algoritmo de VRP, para que a posteriori un usuario del framework (desarrollador de aplicaciones) pueda derivar un algoritmo específico para su problema concreto.
Como se estudió en el capítulo 3, existen tres formas de resolver un problema de ruteo de vehículos: con métodos exactos, heurísticas y metaheurísticas. Las

heurísticas son el enfoque más estudiado y utilizado, lo que permite más fácilmente identificar una estructura genérica, cosa que no se da con los otros dos mecanismos.

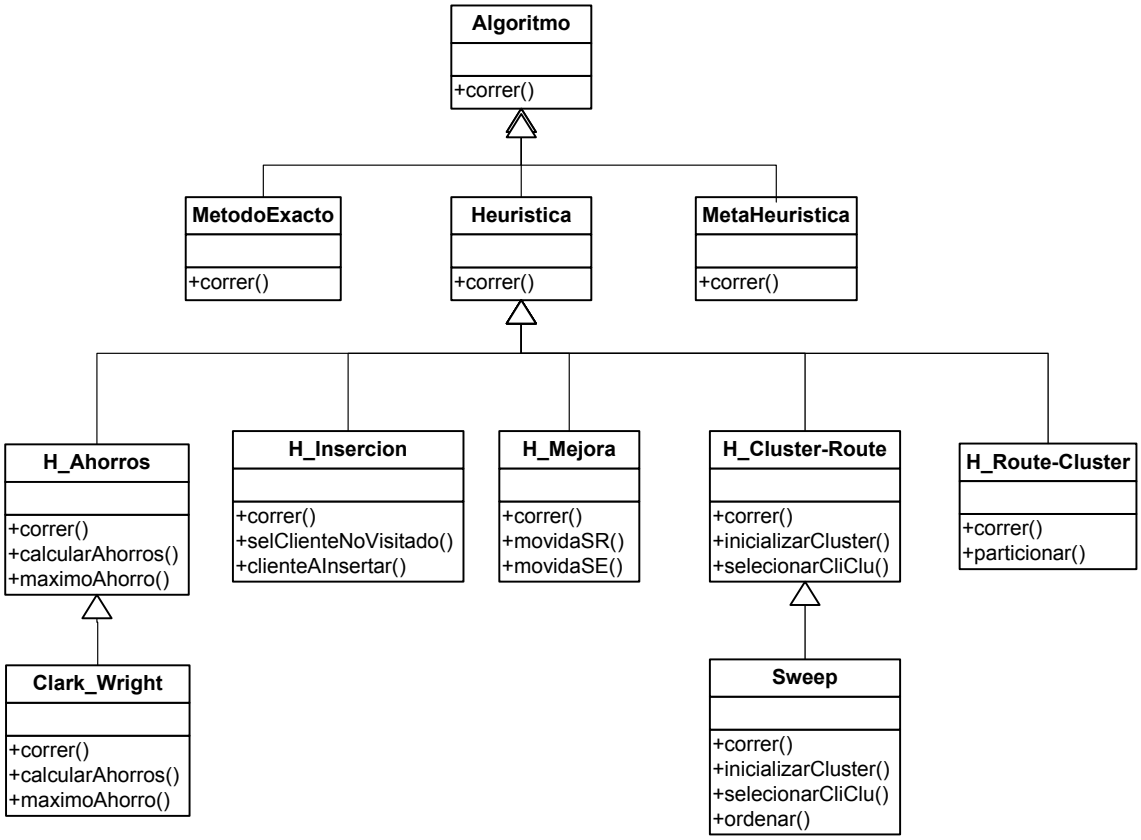


Figura 16: Modelo de Métodos de Solución

Al estudiar la bibliografía se pudo encontrar que existen 5 grandes tipos de heurísticas para resolver un VRP [23]:

- Ahorro
- Inserción
- Cluster First – Route Second
- Route First – Cluster Second
- Mejoramiento

Es así que el modelo de datos que representa las formas de solucionar un VRP es el de la Figura 16.

7.1 Heurísticas de Ahorro

La idea central de las Heurísticas de Ahorro es que se van combinando las rutas, en la medida que al pasar a ser una sola ruta se producen ahorros de costos. Uno de los algoritmos de este tipo, y a la vez de los más difundidos para el VRP es el Algoritmo de Ahorros de Clarke & Wright. Si en una solución dos rutas diferentes (0 , ... , i , 0) y (0 , j , ... , 0) pueden ser combinadas formando una nueva ruta (0 , ... , i , j , ... , 0) como se muestra en la Figura 17, el ahorro en distancia obtenido por dicha unión es:

$S_{ij} = C_{io} + C_{oj} - C_{ij}$ pues en la nueva solución los arcos (i , 0) y (0 , j) no serán utilizados y se agregará el arco (i , j). En este algoritmo se parte de una solución inicial y se realizan las uniones que den mayores ahorros, siempre que no violen las restricciones del problema.

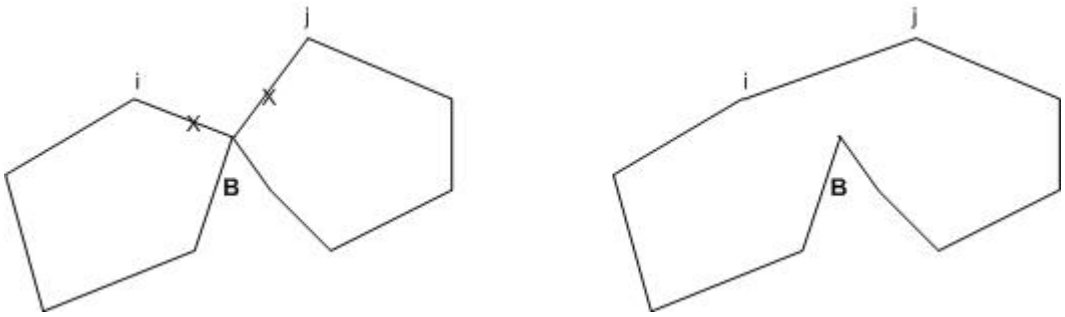


Figura 17: Ejemplo para el Algoritmo de Ahorros de Clark & Wright

Algoritmo de Ahorros de Clark & Wright

Paso 1:

Para cada cliente i construir la ruta (0 , i , 0).

Paso 2:

Calcular $S_{ij} = C_{io} + C_{oj} - C_{ij}$ para todo i y j.

Paso 3:

Sea $S_{i^*j^*} = Max\, S_{ij}$, donde el máximo se toma entre los ahorros que no han sido considerados aún. Sean r_{i^*} y r_{j^*} las rutas que contienen a i* y j*

respectivamente. Si i* es el último cliente de r_{i*} y j* el primer cliente de r_{j*}, y la combinación es factible (capacidad, máxima distancia, etc), combinarlas. Si quedan ahorros por examinar ir a 3.

Una de las deficiencias del algoritmo de Clark & Wright es que tiende a producir buenas rutas al comienzo, pero rutas menos interesantes hacia el final. Para remediar esto, algunos autores [23] han propuesto ahorros de la forma:

$$S_{ij} = C_{io} + C_{oj} - \lambda C_{ij}$$

donde λ es un parámetro que penaliza la unión de rutas con clientes lejanos (llamado shape parameter).

Otros autores [10] plantean que al combinar dos rutas no solamente se considera la posibilidad de insertar una al final de la otra como en el algoritmo de Cl&Wr, sino todas las posibilidades de armar una nueva ruta con los clientes de ambas. De esta forma los ahorros serían de otra forma:

$$S_{ij} = t(S_i) + t(S_j) - t(S_i \cup S_j)$$

Donde S_i es el conjunto de todos los nodos de la ruta que incluye a i, y $t(S_i)$ indica el costo de una solución óptima para el TSP (Problema del Vendedor Viajero) sobre el conjunto de clientes S_i. Con estos cambios, se hace más costoso el cálculo de los ahorros, y además tienen que ser recalculados en cada iteración.

En los algoritmos de ahorro, lo que ocupa bastante tiempo es el calcular el máximo de todos los S_{ij}, por lo que se debe poner cuidado en el método de ordenar que se usará (quicksort, heapsort, etc). Por otro lado podemos lograr ahorros de tiempo y memoria de almacenamiento, si sólo se calcula un subconjunto de todos los S_{ij}, por ejemplo los α más cercanos de cada uno, teniendo así solo $\alpha * n$ ahorros en vez de n^2.

Como se puede observar, los distintos algoritmos de ahorro tienen una estructura común (Figura 18), a partir de la cual se pueden derivar- vía herencia o composición - las actuales y futuras heurísticas de ahorro para el VRP.

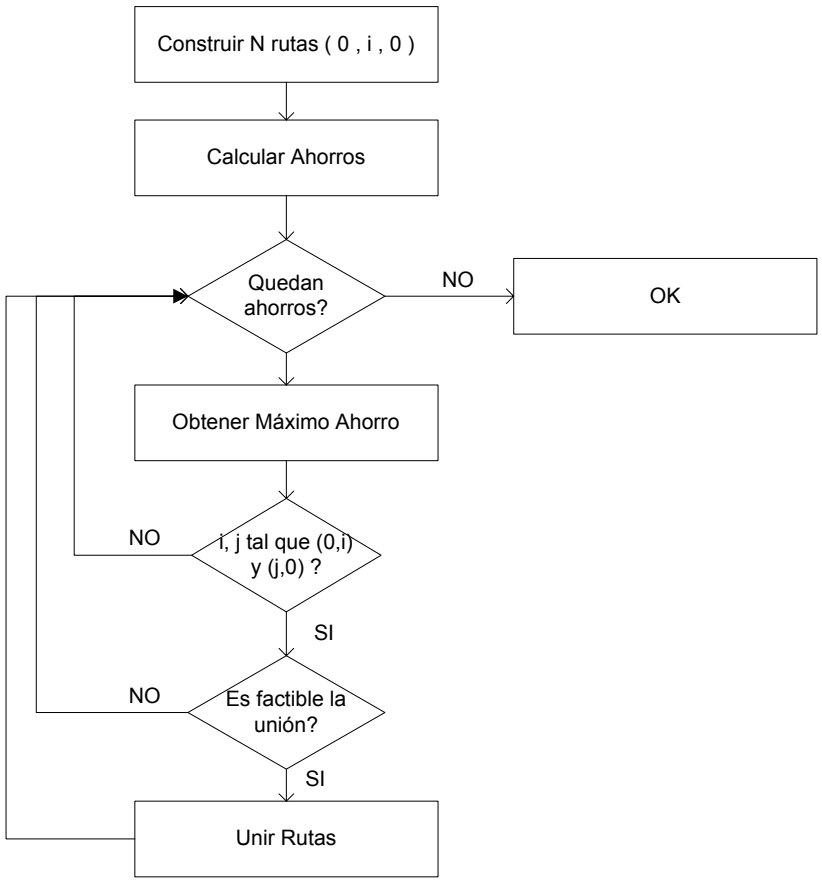


Figura 18: Estructura de algoritmo de Ahorros

En seudo codigo, el diagrama de la Figura 18 es:

```
class H_ahorro extends Heuristica {

   public Double S[][]; // Ahorros
   public Boolean Usado[][]; // Ahorros usados
   public Double lambda; // Parametro en el calculo de ahorros

   // Calcular los ahorros, y dejarlos en S
   public CalcularAhorros();
```

```
// Determina el par (cli_id_i, cli_id_j) que tienen el maximo ahorro
public Double maximoAhorro(cli_id_i , cli_id_j);

// El algoritmo
public planRuteo Correr(vrp_problem p) {

   // Se crea una solucion donde hay n toures, uno para cada cliente
   planRuteo solucion=new planRuteo();
   tour solucion.toures[]=new tour[p.c.cantidad-1];
   for(i=1; i<=p.c.cantidad;i++) {
      Tour Aux_tour=new tour(i, i) ;
      solucion.AgregaTour(aux_tour);
   }

   CalcularAhorros(); // Ahorros se guardan en S

   while (true) {
     maximo_s=maximoAhorro(cli_id_i , cli_id_j);
     if (maximo_s <= 0) break;

     // Marcar ahorro como usado
     Usado[cli_id_i][ cli_id_j]=true;

     tour_i=solucion.buscaTour(cli_id_i);
     tour_j=solucion.buscaTour(cli_id_j);
     if  ( tour_i.terminaEn(cli_id_i) && tour_j.iniciaEn(cli_id_j) ) {
        Tour_union = new Tour();

        // Se unen los toures i y j
        Tour_union = tour_i.union(tour_j);

        if  ( tour_union.esFactible() ) {
        // Si se respetan las restriccs. del tour (largo, capacidad, etc)

        // Se eliminan los dos toures y se crea uno nuevo con la union
           id= Min(tour_i.id, tour_j.id) ;
           Solucion.borrartour(tour_i.id); Solucion.borrartour(tour_j.id);
           Tour_union.id = id;
           Solucion.agregaTour(Tour_union)  }  }
```

Implementemos el algoritmo de Clark & Wright a partir de la clase genérica de algoritmos de ahorro:

```
class ClarkWright extends H_ahorro {

   public CalcularAhorros() {
     for (i =1 ;i<=p.c.cantidad ;i++) {
        for (j =1 ;j<=p.c.cantidad ;j++) {
           S[I][j]= p.m.calcularDistancia(i,0) + p.m.calcularDistancia(0,j) -
p.m.calcularDistancia(i,j);
        }
     }
   }

   public maximoAhorro(arg_i,arg_j) {
     for (i =1 ;i<=p.c.cantidad ;i++) {
        for (j =1 ;j<=p.c.cantidad ;j++) {
           if (S[I][j] >= aux_max_s) {
              aux_max_s=S[I][j];
              arg_i=i; arg_j=j;
           }
        }
     }
   }
}
```

7.2 Heurísticas de Inserción

Las heurísticas de inserción son métodos constructivos en los cuales se crea una solución mediante sucesivas inserciones de clientes en las rutas. En cada iteración se tiene una solución parcial -cuyas rutas sólo visitan un subconjunto

de clientes-, y se selecciona un cliente no visitado para insertarlo en una de las rutas de la solución.

El algoritmo más conocido de este tipo es el de Mole & Jameson [9], en el cual se usan dos medidas para insertar un nuevo cliente en la solución. Para cada cliente no visitado se calcula la mejor ubicación para ubicarlo en la ruta actual. Se tiene la ruta actual ($v_0, v_1,$ ... , v_t, v_{t+1}) donde v_0 = v_{t+1} = 0. El costo de insertar un cliente no visitado w, entre i e i + 1 es:

$$C_1 (v_i , w) = \begin{cases} C_{v_i,w} + C_{w,v_{i+1}} - \lambda C_{v_i,v_{i+1}} & si\ (v_o,...,v_i,w,v_{i+1},...,v_{t+1})\ es\ factible \\ \infty & si\ no \end{cases}$$

La mejor posición para insertarlo está dada por:

$$i(w) = arg\ min\ C_1(v_i, w)$$

Ya que es la posición que presenta el menor costo de inserción (C_1)

Si se utilizara sólo la medida C_1 para decidir el próximo cliente a insertar, es probable que los clientes lejanos a la bodega, no sean considerados sino hasta las últimas iteraciones. Por lo tanto, es necesario utilizar un incentivo adicional para la inserción de clientes lejanos. Así se define:

$$C_2(v_i, w) = \mu C_{o,w} - C_1(v_i, w)$$

En cada iteración se busca el cliente que maximiza C_2 y se lo inserta en la posición dada por el mínimo valor de C_1.

Algoritmo de Mole & Jameson

Paso 1:

Si todos los clientes pertenecen a una ruta, terminar, sino seleccionar un cliente no visitado y crear una nueva ruta r = (0 , w , 0)

Paso 2:

Sea r = (v_0, v_1, ... , v_t, v_{t+1}). Para cada cliente no visitado calcular i(w). Si no hay inserciones factibles ir al Paso 1. Calcular $w^* = \arg\max_w C_2(v_{i(w)}, w)$. Insertar w^* luego de $v_{i(w^*)}$ en la ruta r.

Paso 3:

Aplicar 3-opt sobre r. Ir al paso 2

Existen otros algoritmos de inserción que seleccionan el cliente a insertar de otra manera, o sea no usan el criterio de buscar los clientes más alejados de la bodega. También puede variar la forma de cómo se selecciona la posición donde se insertará este nuevo cliente.

En la Figura 19 se ve la estructura genérica de un algoritmo de inserción.

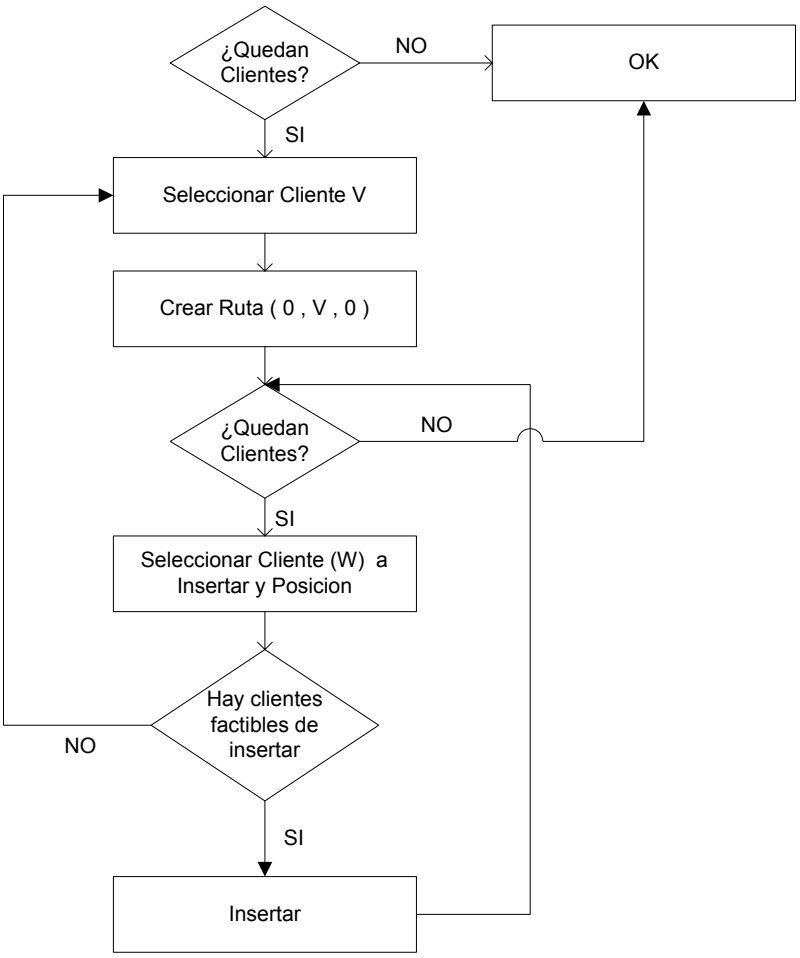


Figura 19: Estructura de Algoritmo de Inserción

Siendo el seudo código fuente:

```
class H_Insercion extends Heuristica {

   // Metodo que selecciona cliente para iniciar una ruta
   Cliente seleccionaClienteNoVisitado();

   // Metodo que selecciona el cliente y posición a insertar
   Cliente clienteAInsertar(int pos);
```

```
   // El algoritmo
   public planRuteo Correr(vrp_problem p) {

      planRuteo solucion=new planRuteo();
      Num_rutas=0;
      quedan=true;
      while (quedan) {
         if ((quedan=!Solucion.todosRuteados())) break;
         Num_rutas++;

         // Se selecciona un cliente no ruteado y se crea una ruta nueva
         Cliente w=seleccionaClienteNoVisitado();
         Tour Actual_tour=new tour(num_rutas , 1 , w.demanda ) ;
         solucion.agregarTour(actual_tour);

         insertar=true;
         while (insertar) {
            // Determina cliente y posición a insertar.
            // Ademas se verifican las restricciones.
            Cliente aux_cliente=clienteAInsertar(donde);
            if ( donde == -1 ) {
               insertar=false;
            }
            else {
               // Se agregar el cliente
              actual_tour.agregarClientePosicion(aux_cliente,donde);
            }
         actual_tour.mejorarsolucion(); // 3-opt por ejemplo }
      }
}
```

7.3 Heurísticas de Asignar Primero, Rutear Después

Los métodos asignar primero y rutear después, proceden en dos fases. Primero se busca generar grupos de clientes, también llamados clusters, que estarán en una misma ruta en la solución final. Luego para cada cluster se crea una ruta que visite a todos los clientes. Las restricciones de capacidad son consideradas en la primera etapa, con la cual en la segunda fase lo que se debe hacer es resolver un problema del vendedor viajero (TSP) para cada cluster.

La heurística más común de este tipo, es la Heurística de Barrido o Sweep, en el cual los clusters se forman girando una semirrecta con origen en la bodega e incorporando los clientes barridos por la semirrecta, hasta que se viole la restricción de capacidad.

Heurística de Barrido o Sweep

Se supone que cada cliente se representa por su ubicación en coordenadas polares: θ y r

Paso 1:

Ordenar los clientes según el ángulo de manera creciente. Seleccionar un cliente w para comenzar y hacer k:=1 y C_k :={w}

Paso 2:

Si todos los clientes pertenecen a algún cluster, ir al paso 3. Si no, seleccionar el siguiente cliente w_i. Si w_i puede ser agregado a C_k sin violar las restricciones de capacidad, hacer C_k := $C_k \cup w_i$. Si no, hacer k := k + 1 y crear un nuevo cluster C_k := w_i. Ir al paso 2.

Paso 3:

Para cada cluster C_k, resolver un TSP.

Otras heurísticas de este tipo, lo que hacen es crear k semillas de clusters, donde los clientes se asocian a un determinado cluster, tras la solución de un problema de Asignación Generalizada (GAP).

En otras heurísticas las k semillas de cluster son elegidas tras la resolución de un problema de localización.

Es así que se obtiene la estructura genérica de un algoritmo de Cluster First – Route Second:

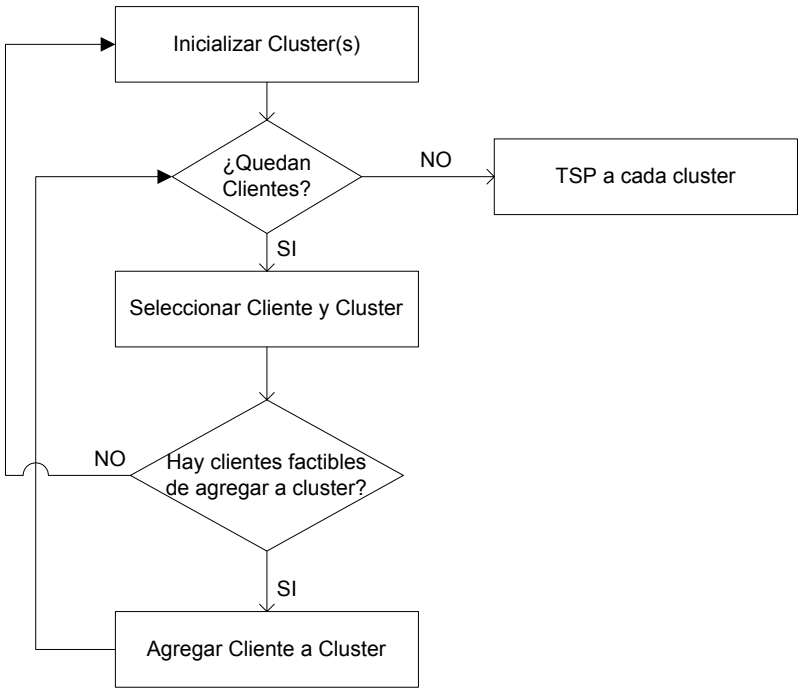


Figura 20: Estructura de Algoritmo de Cluster-First, Route-Second

Siendo el seudo código fuente:

```
Class H_Cluster-Route extends Heuristica {

   Public tour clusters[];
   Public inicializarCluster();
   cliente SeleccionarCliclu();

   // El algoritmo
   public planRuteo Correr(vrp_problem p) {
      planRuteo solucion=new planRuteo();
      num_cluster=0;      quedan=true;
      while (quedan) {
         num_cluster++;   encluster=true;
         inicializarCluster(num_cluster-1);
```

```
         while(encluster){
            // Se selecciona cliente a insertar en cluster
            aux_cliente=seleccionarCliclu();
            aux_tour=clusters[num_cluster-1];
            aux_tour.Agregar(aux_cliente) ;
            if ( !aux_tour.esFactible() ) { encluster=FALSE;}
            else {
               clusters[num_cluster-1].Agregar(aux_cliente);
            }
         }
         quedan=!s.todosRuteados();
      }
      for (i=0;i< num_cluster;i++) {
         // Se rutea cada cluster
         clusters[i].tsp();
      }
      solucion.toures=clusters;
      return solucion;
   }
```

Ahora se deriva el algoritmo Sweep de la clase genérica H_Cluster-Route, implementando métodos que no lo estaban, agregando parámetros y sobrescribiendo métodos.

```
Class Sweep extends H_Cluster-Route {

InicializarCluster(int i) {
   clusters[i]= new tour();
}

cliente seleccionarCliclu {
   int cont=0;
   for(i=1;i<=p.clientes.cantidad;i++) {
      if (!s.estaruteado(i)) {
```

```
            theta[cont]=arctg(y/x);
            radio[cont]=p.getdistancia(0,i);
            id[cont]=i;
         }
      }
      ordenar();
      return(p.clientes.listaclientes[id[0]]);
}
```

7.4 Heurísticas de Rutear Primero, Asignar Después

En los métodos rutear primero – asignar después, también se procede en dos fases. Primero se obtiene una ruta que visita a todos los clientes resolviendo un TSP. Esta ruta la mayoría de las veces no respeta las restricciones del problema, por lo cual se particiona en varias rutas, cada una de las cuales si es factible. Los distintos tipos de algoritmos de este tipo varían en la forma que se obtiene la partición del tour inicial.

El seudo código fuente genérico seria:

```
Class H_Route-Cluster extends Heuristica {

   // El algoritmo
   public planRuteo Correr(vrp_problem p) {
      tour aux_ruta;
      for(i=1;i<=p.clientes.cantidad;i++) {
         aux_ruta.Agregar(p.c.lista_clientes[i]) ;
```

```
        }
        aux_ruta.tsp();
        aux_plan_ruteo=particionar(aux_ruta) ;
        return aux_plan_ruteo;
    }
```

Donde lo importante es definir el método particionar.

7.5 Heurísticas de Mejoramiento

A diferencia de los cuatro métodos anteriores, las heurísticas de mejoramiento no generan un conjunto de rutas, sino que mejoran uno ya obtenido a través de alguno de los métodos ya mostrados.

Dada la solución obtenida, lo que se hace es evaluar las soluciones vecinas de ésta, las cuales se obtienen a través de reglas o procedimientos sencillos –llamadas movidas- al interior de las rutas o entre las rutas.

Uno de los procedimientos más conocidos de movidas al interior de una ruta, es el λ-intercambio, que consiste en eliminar λ arcos de la ruta y reconectar los

segmentos restantes en todas las posibles formas. El procedimiento se detiene en un mínimo local, cuando no se pueden obtener mejores rutas.

También se pueden realizar movidas entre varias rutas, siendo los operadores de Van Breedam los más usados [23]. En el operador String Relocation (SR), una secuencia de m nodos es transferida de una ruta a la otra manteniendo el orden de la ruta original. En el operador String Exchange (SE) una ruta envía una secuencia de m clientes a la otra y esta ultima envía otra secuencia de n clientes a la primera. Simbólicamente se denota con (m,0) a cada SR y con (m,n) a cada SE.

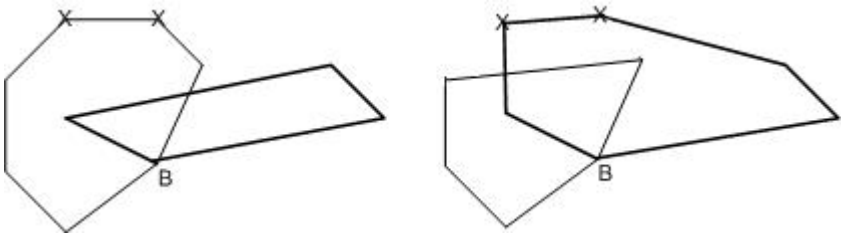


Figura 21: Ejemplo de Movida SR (2,0)

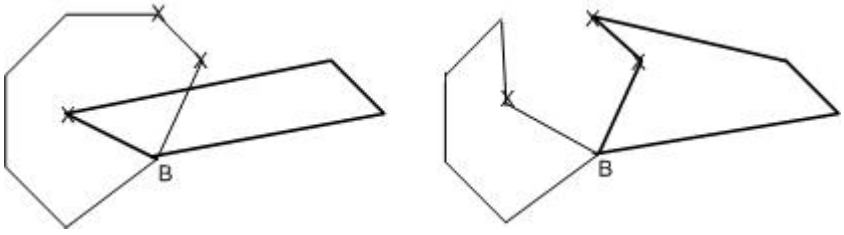


Figura 22: Ejemplo de Movida SE (2,1)

7.6 Operadores básicos y Algoritmos futuros

Como se pudo apreciar, los métodos de solución heurísticos se agruparon en 5 categorías, pero además se pudo observar en la práctica, cuales eran las operaciones básicas que constituían todo método de solución. Estas cuatro operaciones son:

- Crear nueva ruta
- Agregar cliente(s) a una ruta
- Combinar dos rutas
- Intercambiar el orden en una ruta

Estas, se vieron representadas en el código fuente entre otros en los siguientes métodos:

Métodos de la clase PlanRuteo

```
AgregaTour(tour t);
// A la solución actual le agrega un nuevo tour t

tour BuscaTour(int id_cliente);
// Busca el tour en el que está el cliente de identificador id_cliente

borrarTour(int id_tour)
// De la solución actual borra el tour de identificador id_tour
```

```
boolean todosRuteados();
// Verifica si todos los clientes están en una ruta

boolean ruteado(int id_cliente);
// Verifica si el cliente id_cliente esta ruteado en alguno de los tours del
Plan de Ruteo

tour buscaTourCliente(int id_cliente)
// Busca el tour en el cual está el cliente id_cliente

tour mezcla(tour t1, int cli1, tour t2, int cli2)
// Junta los toures t1, t2, uniendo los clientes cli1 y cli2
```

Métodos del objeto Tour

```
tour(int id_tour, int id_cliente)
// Constructor . Crea un nuevo tour de identificador id_tour, cuyo primer
// cliente es el de identificador id_cliente. Además setea el numero de
// clientes en 1, y la demanda del tour como la demanda del cliente id_cliente

agregar(cliente c);
// Agrega el cliente c al Tour

agregarClientePosicion(cliente c, int donde);
// Agrega al tour el cliente c, en la posición donde

Tsp();
// Dado un tour, resuelve el problema del vendedor viajero
```

A partir de estos métodos, se hace más fácil la implementación de nuevos algoritmos que resuelvan el VRP, los cuales no necesariamente se clasifiquen en algunos de los 5 tipos estudiados. O sea estamos dando las herramientas a los usuarios del framework (desarrollador de aplicaciones) para que implementen los nuevos algoritmos que vayan apareciendo.

8. Resultados

Como se mencionó en el capítulo 3, la mayoría de los frameworks se desarrollan con el enfoque orientado a objeto, ya que permite una instanciación de forma más clara. Es por eso que el framework y el software final se hicieron en un lenguaje como Java, en específico se programó en Jbuilder 7.0.

En la Figura 23 se observa un pantallazo del software, en el cual se muestra el problema ya resuelto, mostrando la solución en el mapa, y también como reporte.

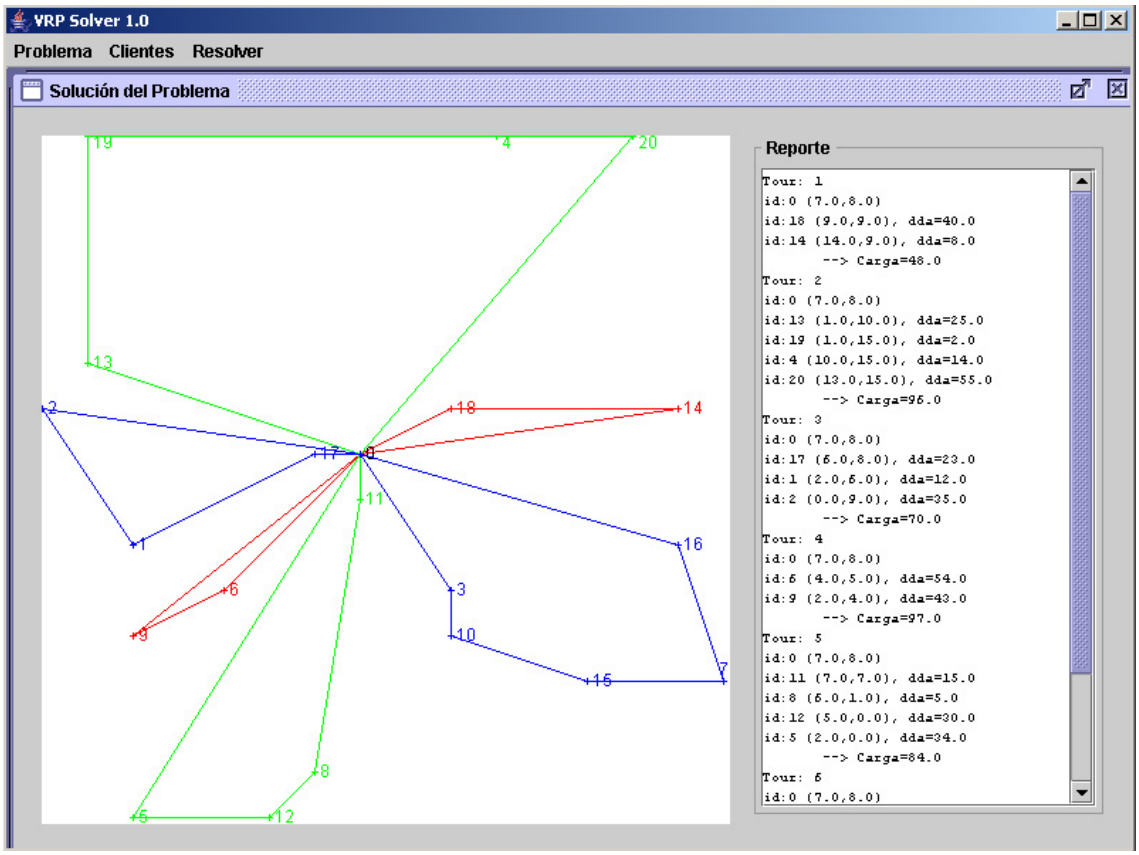


Figura 23: Software de Ruteo de Vehículos

También corresponde evaluar la calidad del software final en lo que se refiere a la velocidad con que se obtienen las soluciones para distintos problemas, ya que podemos haber obtenido un excelente framework pero los software que se derivan de él no tienen una buena performance.

Num. Clientes	Clark & Wright			Sweep		
	Tiempo (segs.) (1)	Toures	Distancia (kms.)	Tiempo (1)	Toures	Distancia
25	0.5	5	364.34	0.3	5	282.35
50	1.0	10	576.35	0.6	10	510.51
100	2.9	20	1271.48	1.3	22	1213.51
200	7.6	34	2492.96	1.6	38	1723.02

Num. Clientes	Sweep con Ventanas de Tiempo				
	Tiempo (segs.) (1)	Toures	Distancia (kms.)	Tiempo Recorrido (hrs.)	Promedio por Tour (hrs.)
25	0.6	7	344.92	80.17	11.45
50	1.0	12	614.86	132.26	11.02
100	1.9	26	1524.91	266.06	10.23
200	3.3	53	2538.30	526.67	9.94

(1) Problemas corridos en un Pentium IV de 3.0 Ghz, 512 MB RAM, con Windows XP

Las pruebas se realizaron con problemas generados aleatoriamente (coordenadas, demanda, y ventanas) en un área de 30x30 kms.

Por otra parte se quiso hacer una comparación con resultados obtenidos a partir de instancias predeterminadas, en este caso las de Van Breedam [32], las cuales tienen 100 clientes, la demanda de estos es 10, la zona geográfica es de 100 x 100 kms. y varían en la capacidad de los camiones. Obteniéndose los siguientes resultados al comparar el metodo “Sweep” y la mejor solución conocida según [1].

Instancia	
Instancia	Capacidad Camiones
Bre-2	100
Bre-4	50
Bre-6	200
Bre-9	100

Sweep		
Tiempo (segs.)	Toures	Distancia (kms.)
1.7	10	1643.1
1.4	20	2294.4
1.8	5	1097.0
1.2	10	1993.9

Mejor Resultado
Distancia (kms.)
1506.0
1470.0
969.0
1690.0

8.1 Ejemplos de Instanciaciones del Framework

Lo más importante de un framework es que este sea fácilmente instanciado a distintas aplicaciones específicas. Para eso acá se verán tres problemas que mostrarán cómo hacerlo.

Problema A

- Se contará con varias ventanas (número variable) de tiempo
- Los servicios tendrán una duración determinada e igual para todos
- Los productos a transportar son varios
- La posición de los clientes estará dada por la dirección

Un ejemplo de este problema son los despachos a domicilio de los supermercados (telemercados)

Para resolverlo debemos proceder como sigue.

1) El archivo de datos tendrá la siguiente estructura:

```
%PROBLEMA
VENTANAS=N//1,0  # Cantidad variable de ventanas
DEMANDA=SI/200/1,0   # Numero de productos
VOLUMEN=SI//1,1   # Tamaño en m3 de la carga
DURACION=SI/10/0,0  # El despacho en promedio toma 10 minutos
FLOTA=HOMOGENEA//0,0
POSICION=DIRECCION//0,0
CANTIDAD_VEH_TIPO=SI//0,1
%CLIENTES
Ricardo Lyon 341, Providencia # La bodega
Hendaya 3408, Las Condes;VENTANAS=08:00-12:00;14:00-18:00;DEMANDA=2,0,0,1,0..;VOLUMEN=10
Lo Plaza 1207 Dpto 305, Ñuñoa; VENTANAS=10:00-17:00;DEMANDA=1,1,0,0,0,0,0,2..;VOLUMEN=75
.....
%VEHICULOS
Camion Standard;CANTIDAD_VEH_TIPO=10;VOLUMEN=200
```

2) Los cambios e incorporaciones que habría que hacer en el código son:

i) Diseñar un algoritmo que resuelva este tipo de problema específico

```
algoritmo_n_ventanas a1 = new algoritmo_n_ventanas();
a1.cargaCaracteristicas();
lista_algoritmos.addElement(a1);

algoritmo_n_ventanas extends heuristicaClusterFirst {
   public void cargaCaracteristicas() {
      creaCaracteristica("FLOTA","HOMOGENEA");
      creaCaracteristica("VENTANAS","N");
      creaCaracteristica("DEMANDA","SI");
      creaCaracteristica("VOLUMEN","SI");
      creaCaracteristica("DURACION","SI");
      creaCaracteristica("POSICION","DIRECCION");
      creaCaracteristica("CANTIDAD_VEH_TIPO","SI");
}
}
```

ii) Para poder aceptar que los pedidos y los vehículos tengan un volumen se debe agregar a los métodos verificaConsistenciaGuardaVehiculo y verificaConsistenciaGuardaCliente de la clase caracteristica:

```
else if (nombre.compareTo("VOLUMEN") == 0) {
   float vol= new Float(par_caract).floatValue();
((tipo_vehiculo)p.f.lista_tipo_vehiculos.elementAt(cont_id)).cap.setVolumen(vol);
}
```

iii) Para indicar que todas las visitas tienen la misma duración se debe modificar el metodo verificaConsistenciaGuardaCliente de la clase caracteristica:

```
else if (nombre.compareTo("DURACION") == 0) {
      ((cliente)p.c.lista_clientes.elementAt(cont_id)).duracion=subvalor;
}
```

iv) Para permitir que las posiciones de los clientes sean direcciones se debe agregar al método verificaConsistenciaGuardaCliente de la clase caracteristica:

```
else if (nombre.compareTo("POSICION") == 0) {
if (valor.compareTo("Direccion") == 0 ) {
((cliente)p.c.lista_clientes.elementAt(cont_id)).pos.setCoordenadas(par_caract);
}
}
```

y también agregar a la clase posición un método que traduzca las direcciones en coordenadas (x,y).

```
setCoordenadas(string s) {
# Leer desde una Base de Datos Geografica a que posición (x,y) corresponde una
# direccion.
}
```

v) Para el hecho de que el número de ventanas es variable, no hay que hacer nada, porque el framework está diseñado para recibir un número variable de ventanas.

Problema B

- Habrá un máximo tiempo de ruteo
- No habrá ventanas de tiempo
- La posición y distancia entre clientes está fija

Un ejemplo de este problema es el transporte de pasajeros de una empresa, en el cual los empleados no pueden llegar atrasados al trabajo. Además los paraderos son siempre los mismos.

Para resolverlo debemos proceder como sigue.

1) El archivo de datos tendrá la siguiente estructura:

```
%PROBLEMA
DEMANDA=SI/1/1,0   # Solo hay un tipo de elemento a transportar: los empleados
FLOTA=HOMOGENEA//0,0
CANTIDAD_VEH_TIPO=SI//0,1
MXO_TIEMPO_RUTEO=SI/120/0,0
%CLIENTES
15,7
10,40;DEMANDA=7
7,90;DEMANDA=5
......
%VEHICULOS
Camion Standard;CANTIDAD_VEH_TIPO=10;DEMANDA=45
```

2) Los cambios e incorporaciones que habría que hacer en el código son:

i) Ocupar alguno de los algoritmos existentes ya que es un problema clásico, a excepción de la restricción del máximo ruteo (modificar clase esFactible)

ii) Como se decía, los paraderos son siempre los mismos, por lo cual ya está calculado cual es su posición y la distancia entre ellos, solo hay que leerlos desde un archivo. Para eso debemos modificar el método obtenerDistancias del objeto matrizClientes:

```
public void obtenerDistancias(clientes c) {
   FileInputStream  fis=new FileInputStream((File)fc.getSelectedFile());
   BufferedReader d= new BufferedReader(new InputStreamReader(fis));
   Double costo;
   String lista1[];
   String lista2[];
   int i,j; # Formato del archivo:  2,7;30.12
   while ((nextLine = d.readLine()) != null){
      lista1=nextLine.split(";");
      lista2=lista1[0].split(",");
      i=new Integer(lista2[0]).intValue();
      j=new Integer(lista2[1]).intValue();
      costo=new Double(lista1[1]).doubleValue();
      celda[i][j].costo=costo;
   }
}
```

Problema C

- Habrá más de una bodega
- La flota será heterogénea
- No hay restricciones de capacidad ni en bodega ni vehículos
- Habrán despachos y recolecciones combinadas
- Habrá un máximo tiempo de ruteo
- Los vehículos pertenecen a las bodegas, o sea deben volver a la misma de la que partieron.

Un ejemplo de este problema podría ser un servicio de correo o pequeñas encomiendas. O también podría ser el caso de servicios a domicilio (ejemplo: servicios técnicos de telefonía)

Para resolverlo se debe proceder como sigue.

1) El archivo de datos tendrá la siguiente estructura:

```
%PROBLEMA
MULT_BODEGAS=SI/3/0,0
FLOTA=HETEROGENEA//0,0
CANTIDAD_VEH_TIPO=SI//0,1
MXO_TIEMPO_RUTEO=SI/120/0,0
PERTENECE_BODEGA=SI//0,1 # Los vehículos pertenecen a bodegas
DURACION=SI/5/0,0 # Los servicios tienen un tpo fijo para todos los clientes
%CLIENTES
15,7
10,40
15,23
15,35 # Primer Cliente
20,12
......
%VEHICULOS
Camion Standard;CANTIDAD_VEH_TIPO=10;PERTENECE_BODEGA=1
Camioneta; CANTIDAD_VEH_TIPO=5;PERTENECE_BODEGA=2
Camion Standard;CANTIDAD_VEH_TIPO=3;PERTENECE_BODEGA=3
....
```

2) Los cambios e incorporaciones que habría que hacer en el código son:

i) Diseñar un algoritmo que resuelva el problema. Para eso se podría proceder así:

- Asignar clientes a bodegas, según criterio geográfico.
- Para cada bodega aplicar un algoritmo de TSP, modificado para que permita máximo tiempo de ruteo.

ii) Para poder permitir varias bodegas habría que modificar el diagrama de clases de la siguiente manera:

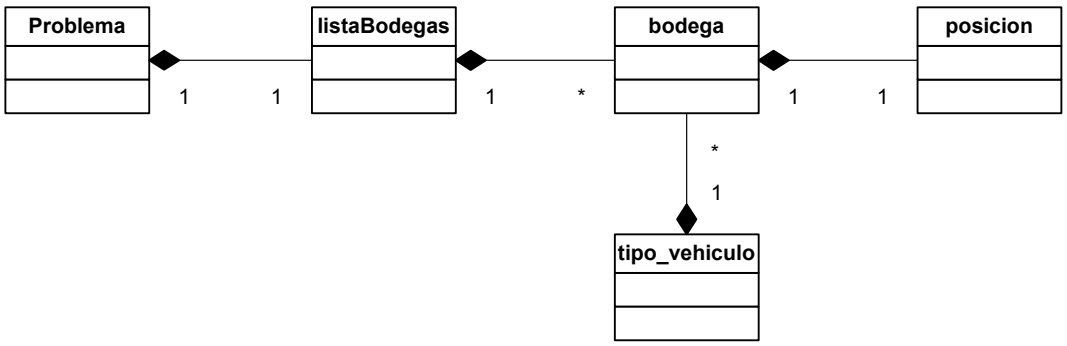


Figura 24: Modificación del Modelo para múltiples bodegas

iii) Para indicar que todas las visitas tienen la misma duración se debe modificar el metodo verificaConsistenciaGuardaCliente de la clase caracteristica:

```
else if (nombre.compareTo("DURACION") == 0) {
      ((cliente)p.c.lista_clientes.elementAt(cont_id)).duracion=subvalor;
}
```

iv) Para poder aceptar que hayan mútiples bodegas se debe agregar al método verificaConsistenciaGuardaCliente de la clase caracteristica:

```
else if (nombre.compareTo("MULT_BODEGAS") == 0) {
   P.listabodegas.cantidad=subvalor;
}
```

Además hay que modificar el método leeclienteArchivo para que considere a los p.listabodegas.cantidad primeros clientes, como bodegas.

v) Para permitir que los tipos de vehículos pertenezcan a las bodegas se debe agregar al método verificaConsistenciaGuardaVehiculo de la clase caracteristica:

```
else if (nombre.compareTo("PERTENECE_BODEGA") == 0) {
   p.tipo_vehiculo.bodega=subvalor;
}
```

8.2 Análisis de Resultados

8.2.1 Propiedades del Framework

El framework que se obtuvo cumple algunas de las propiedades deseables mencionadas en 3.1.2.4

Es **fácil de usar** ya que lo que el usuario debe escribir/sobrescribir es bastante limitado, que son los métodos de solución y el formato del archivo de datos, todo el resto (diagrama de clases, interfaz gráfica, etc) prácticamente no hay que tocarlo.

Es **extensible**, ya que se pueden agregar nuevas componentes, por ejemplo en lo que se refiere a: reportes de las soluciones; a la interfaz grafica (modificando solo una clase); a las ubicaciones de los clientes, que podrían ser extraídas de un sistema de información geográfico (a través del hotspot 'posicion'); a poder agregar mas restricciones propias de los VRP, como podría ser tener más de una bodega, lo cual no sería tan difícil de modificar en el diagrama de clases. El **problema de la instanciación C** nos muestra que no es tan complejo poder ir un poco más allá del dominio definido.

En lo que se refiere a **ser flexible** no lo es, ya que es difícil encontrar un dominio similar al de ruteo de vehículos; podría ser al de scheduling de tareas,

pero habría resultado un framework demasiado genérico y que iría en perjuicio de la **facilidad de usar**.

Se podría decir que cumple la propiedad de **completitud**, ya que cubre el dominio que se especificó (una bodega, demanda en los nodos, un vehículo por cliente) gracias al mecanismo de 6.3. Los **problemas de instanciación A y B** nos muestran como abarcar el dominio que se definió y lo relativamente sencillo que es llevarlo a una aplicación específica. Sin embargo no se pudieron determinar mecanismos o métricas para determinar la completitud.

8.2.2 Análisis Costo Beneficio

Como se indicó en el capítulo 3, el reuso de software -y en específico el uso de frameworks- permite que a partir de ellos se reutilicen activos de software (como código fuente, esquemas de bases de datos, etc) lo que conlleva una mejora de la productividad, ya que se reducen los tiempos de desarrollo, lo que trae un ahorro de costos. Pero desarrollar un framework también trae costos, como son por ejemplo: hacer un análisis de requerimientos más amplio y exhaustivo, hacer un diseño más genérico, etc. es por eso que se necesita hacer un análisis costo/beneficio.

Sea RCR el Costo Relativo de Reuso [27], que es la porción de esfuerzo que toma reusar un componente sin modificación, versus escribirlo desde el comienzo. El RCR puede variar dependiendo de muchos factores y su valor va por lo general entre 0.03 y 0.4. [27] recomienda un RCR de 0.2, es decir que cuesta 20% de esfuerzo reusar software de lo que cuesta escribirlo del comienzo. El RCR tiene relación con la complejidad del componente de software que se esta reusando, a mayor complejidad, mayor RCR.

Por otro lado sea el RCRW el Costo Relativo de escribir para Reuso [27], que es la porción de esfuerzo que toma escribir un componente reusable versus escribirlo para usarlo por una única vez.

RCWR = Esfuerzo de desarrollar para reuso/Esfuerzo para desarrollar sin reuso

El RCWR puede variar dependiendo de muchos factores, entre 1.0 y 2.2. [27] recomienda 1.5, esto significa que cuesta un 50% de esfuerzo adicional escribir softare reusable. A mayor complejidad, mayor RCWR.

Usando ambas métricas se puede llegar a lo propuesto por [15]:

$$C = (1 - R) + (RCR + \frac{RCWR}{n}) * R$$

donde:

C: Costo relativo de un proyecto (0 <= C <= 1)

R: Porción de código reusado en el proyecto (0<=R<=1)

n: número de veces sobre el cual los desarrollos serán amortizados

Ya que no es tan complejo nuestro framework, asignemos los valores propuestos por [27] y supongamos que dado que el código a escribir principalmente es el método a utilizar, y el formato de los datos, R toma el valor de 0.7. Así:

$$C = (1 - 0.7) + (0.2 + \frac{1.5}{n}) * 0.7$$

$$C = 0.44 + \frac{1.05}{n}$$

Ahora integramos para obtener cuanto es el costo acumulado cuando se va reutilizando el framework, osea cuando va creciendo n.

$$C = 0.44 * n + 1.05 \ln(n) + K$$

Sea la constante de integración K=1 que viene a representar el costo inicial de desarrollo del framework, y ya que RCWR es un costo relativo.

Es así que la comparación entre desarrollar software adhoc y a partir de frameworks queda esquematizada por el siguiente grafico:

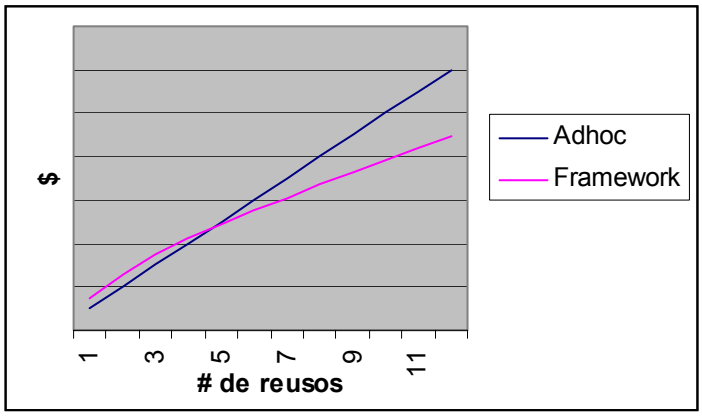


Figura 25: Comparación de desarrollar con y sin framework

Del gráfico podemos concluir que a partir de aproximadamente el 5º uso ya sale rentable (con los parámetros que se usaron) usar el esquema de frameworks por sobre el de hacer software adhoc.

9. Comentarios y Conclusiones

En el presente trabajo se desarrolló un framework orientado a objeto para el problema de ruteo de vehículos, el cual permite que a partir de él se desarrollen nuevos softwares sin mayor análisis del dominio.

Un framework es un tipo de arquitectura de software reusable que comprende, no sólo reutilización de código (como las librerías), sino también de diseño. Un framework es un diseño e implementación parcial de un determinado dominio, a partir del cual un usuario desarrolla una aplicación para el problema (perteneciente al dominio) específico que desea resolver. Los frameworks son útiles cuando hay muchas aplicaciones distintas para un mismo dominio, evitando así etapas del desarrollo como son el análisis de requerimientos, logrando así ahorros de tiempo/recursos.

Los Frameworks generan aplicaciones para un dominio entero, por lo tanto deben haber puntos de flexibilidad que se puedan modificar de acuerdo a los requisitos particulares de la aplicación. A estos puntos se les llama **hotspots**, que son las partes variables de una aplicación a otra del dominio de estudio.

El framework en estudio se construyó teniendo como base la metodología de desarrollo de software orientado a objeto, pero con la diferencia que el análisis del problema tiene que hacerse con mayor abstracción para así cubrir todos los problemas del dominio de ruteo de vehículos. Por otro lado también hay que preocuparse que los hotspots sean de verdad puntos de flexibilidad entre uno y otro problema. Otra etapa de la metodología es la de instanciación y testeo, donde se debe demostrar que el framework logró la abstracción necesaria para cubrir el dominio que se autoimpuso.

El problema de ruteo de vehículos es uno de los problemas clásicos de la gestión de operaciones, que en los últimos 50 años ha sido materia de profundo estudio. En esta tesis se logró condensar en el análisis y diseño del framework, elementos que forman parte del problema: clientes, flota, bodega, ventanas, malla, máximo tiempo de viaje, función a optimizar, tipos de ventanas, etc.

El dominio que se estableció es el de ruteo de vehículos con una sola bodega, y todo el resto de las características relativamente libres: múltiples ventanas, múltiples productos, red bidireccionada, penalización por no visita o atraso, etc. En todo caso, problemas específicos como el SDVRP y VRPPD no entran en el dominio de estudio.

Por otro lado, se estudiaron los software de ruteo de vehículos, que aparte de resolver un VRP tienen otras características como son: ingreso de datos, visualización gráfica de la solución, reporte con la solución, almacenamiento de la solución, comparación de soluciones, entre otros.

Como se dijo, la forma de desarrollar el framework fue basándose en el esquema típico de desarrollo OO. Es así que en la etapa de análisis se identificaron tres grandes caso de usos: ingreso de datos, resolución del problema, modificación del problema. También se desarrolló un modelo conceptual donde se muestran los elementos de la vida real que interactúan. Para finalmente obtener el diagrama de actividades del framework, en el cual destaca el mapeo del problema, que se refiere a asignar a un problema en especifico un algoritmo en específico que lo resuelva.

En el análisis también es importante detallar cada uno de los hotspots que en este caso se determinaron que eran cuatro: calculo de la malla, capacidad, tipos de vehículos y posición. Podrían ser más pero estos eran los principales.

Pero el principal de los hotspots era poder abstraer los mecanismos de solución, o sea el como estos obtienen la solución. Se clasifican en tres: métodos exactos, heurísticas y metaheurísticas. Este estudio se concentró en las heurísticas por ser las más usadas, las que obtienen buenas soluciones en

relación con el tiempo que toman, y las más fáciles de entender. Se estudiaron cinco tipos de heurísticas, logrando obtener la arquitectura base de cada una de ellas, o sea como lo hacen para resolver el VRP.

En la etapa de diseño se logró avanzar desde el modelo conceptual hacia el modelo de clases, obteniéndose un modelo que da cuentas de la complejidad del dominio de estudio. Por otro lado se diseñó un mecanismo de ingreso de datos a través de archivos en el cual van: las características del problema, los clientes y los vehículos. Esta forma de ingreso de datos permitió tener un mecanismo de mapeo entre problemas y algoritmos.

Es así que se obtiene un framework y un software, desarrollado en un lenguaje de programación orientado a objetos (Java) que permite usarlo en cualquier plataforma.

El framework obtenido resultó ser de caja blanca y caja negra. Caja blanca porque es necesario escribir/sobrescribir algunos métodos de las clases, para lo cual hay que meterse al código fuente (Java). Caja negra por el mecanismo descrito en 6.3, donde a través de la configuración de parámetros se pueden resolver distintos tipos de problemas (siempre que exista un algoritmo para el problema específico).

En cuanto a los métodos de solución no queda del todo claro, que a partir de los esquemas del capítulo 7, se pueda derivar a través de herencia nuevos métodos de solución (por dificultades al nivel de programación), por ejemplo el algoritmo de ahorros en sus versiones secuencial y paralela presentan diferencias que hacen dificultoso derivarlas a ambas de la clase h_ahorros. Pero por otro lado es muy útil que se hayan identificado las operaciones/metodos básicos que componen un método genérico de resolución de VRPs, ya que permite crear nuevos métodos que incluso no caigan en ninguno de los 5 tipos de heurísticas.

Uno de los resultados esperados era poder obtener una métrica para evaluar la completitud del framework, cosa que dada la complejidad del dominio no se pudo obtener, sólo se instuye (por el mecanismo de 6.3) que el dominio (una bodega, ventanas, un vehículo por cliente, etc) está cubierto con el framework obtenido.

Dado el mecanismo para mapear problemas/algoritmos, puede suceder que pequeñas e insignificantes modificaciones en la estructura del problema haga que no tenga un método de solución apropiado.

En lo que se refiere a documentación del framework, esta tesis es una excelente fuente de información para que los posibles usuarios deriven de ella

aplicaciones específicas para sus propios VRPs. Por supuesto el código fuente del software final es un buen input para los desarrolladores.

El framework desarrollado permitirá a sus usuarios desarrollar software de ruteo de vehículos a un más bajo costo, a un menor tiempo y con una mejor calidad, principalmente porque se hizo un profundo análisis del dominio, que no deberá ser hecho nuevamente. Asimismo en trabajos futuros, el framework obtenido podría ser extendido para trabajar con problemas más complejos (múltiples bodegas, multiperíodos, etc).

El desarrollo de este trabajo ayudó a mostrar que el enfoque de framework es bastante prometedor en lo que se refiere al diseño de componentes de software reusables para problemas complejos de gestión de operaciones, destacando como logros concretos de este estudio:

- Desarrollo de un completo diagrama de clases para el problema de ruteo de vehículos, que condensa todos los elementos que participan en el VRP.

- Estudio de los 5 tipos de heurísticas que existen, y desarrollo de esquemas genéricos de cómo funcionan algorítmicamente cada una de ellas.

- Identificar las operaciones básicas que tiene cualquier método de resolución del VRP, como son: crear ruta, unir rutas, buscar cliente, etc

- Desarrollo de un mecanismo de mapeo entre los problemas y los métodos de solución

- Realización de un análisis costo/beneficio entre desarrollar software adhoc y desarrollarlo a partir de un framework

9.1 Trabajo Futuro

1. El más importante trabajo futuro que debiera seguir a esta tesis es comprobar las características del framework en lo que se refiere a **facilidad de uso** y **completitud**. Para eso se debieran hacer pruebas con distintos desarrolladores (por separado) que implementen a partir de este framework aplicaciones específicas.

2. Verificar que a partir del framework se obtienen soluciones igualmente buenas a las obtenidas con softwares específicos, en lo que se refiere a tiempo de corrida y distancia recorrida. Ya que puede ser que el

framework y su diagrama de clases rigidice la implementación de nuevas heurísticas, obteniendo peores resultados.

3. Desarrollar el mismo estudio que se hizo acá – para heurísticas – para otras formas de solución como metaheurísticas o métodos exactos.

4. Aplicar la teoría de frameworks a otras áreas de la gestión de operaciones como podría ser: asignación de tripulación, ubicación de instalaciones, etc.

5. Desarrollar una mirada más comercial para el framework, por ejemplo mejorando la interfaz gráfica.

10. Bibliografía

1. Alba E. y Dorronsoro B. 2005. New Best-So-Far Solutions for van Breedam's benchmark (Capacited VRP). Department of Computer Science, University of Malaga, España.

2. ArcLogisitics. 2004. ArcLogistics Route: A Complete Routing and Scheduling Solution.

3. Barros, O. 2002. Componentes de Lógica del Negocio desarrollados a partir de patrones de procesos. Documentos de Trabajo: Serie Gestión #34.

4. Butler G. y Denommee P. 1999. Documenting Frameworks. En: Fayad M. (ed.). Building Application Frameworks. New York. John Wiley ans Sons. pp. 495-503

5. Bodin L., Golden B., Assad A. y Ball M. 1983. The State of the Art in the Routing and Scheduling of Vehicles and Crews. Computers and Operations Research 10 (2): 63-211.

6. Bodin L., y Golden B. 1981. Classification in vehicle routing and scheduling. Networks 11: 97-108.

7. Bodin L. 1981. The state of the art in the routing and scheduling of vehicles and crews. 365p.

8. Bodin L. 1990. Twenty Years of Routing and Scheduling. Operations Research 38 (4): 571-579.

9. Cuzmar P. 1992. Estudio y Aplicación de Metodologías de asignación y ruteo vehicular. Memoria de Ingeniero Civil Industrial. Santiago, Universidad de Chile, Facultad de Ciencias Físicas y Matemáticas. 53p.

10. Desrochers M. y Verhoog T. 1989. A matching based savings algorithm for the vehicle routing problem. Les Cahiers du GERAD, Ecole des HautesEtudes Commerciales de Montreal.

11. Eibl P. 1996. Computerized Vehicle Routing and Scheduling in Road Transport. England. Avebury. 318p.

12. Fayad M. y Johnson R. (eds.). 2000. Domain-Specific Application Frameworks. John Wiley and Sons. 704p.

13. Fayad M., Schmidt D. y Johnson R. (eds.). 2000. Building Application Frameworks. John Wiley and Sons. 688p.

14. Froehlich G., Hoover J., Liu L. y Sorenson P. 1998. Designing Object-Oriented Frameworks. Handbook of Object Technology, CRC Press.

15. Guffney J. y Durek T. 1989. Software reuse - key to enhanced productivity: some quantitative models. Information and Software Technology 31 (5): 258-267.

16. Golden B. y Assad A. 1986. Perspectives on Vehicle Routing: Exciting New Developments. Operations Research 34 (5): 803-810.

17. Golden B. y Assad A. (eds.). 1988. Vehicle Routing: Methods and Studies. North-Holland. 480p.

18. Gómez O. 2005. Bases conceptuales de frameworks y su importancia para lograr reutilización. Tertulias de Ingeniería de Software. Universidad EAFIT.

19. Hall R. 2004. Change of Direction: Vehicle Routing Software Survey. OR/MS Today 31 (3).

20. Jacobs-Blecha C. y Goetschalckx M. 1992. The vehicle routing problem with backhauls: properties and solution algorithms. En: National Transportation Research Board: 13 al 15 de Enero 1992. Washington DC.

21. Larman C. 1999. Uml y Patrones: Introducción al análisis y diseño orientado a objetos. México. Prentice Hall. 536p.

22. Laporte M., Gendreau M., , Potvin J-Y. y Semet F. 2000. Classical and Modern Heuristics for the vehicle routing problem. International Transaction in Operational Research 7: 285-300.

23. Laporte G. y Semet F. 1998. Classical Heuristics for the Vehicle Routing Problem. Les Cahiers du GERAD, G98-54, Group for Research in Decision Analysis, Montreal, Canada.

24. Levine D. y Schmidt D. 2003. Introduction to Patterns and Frameworks. Department of Computer Science, Washington University.

25. Mattsson M. 1996. Object-Oriented Frameworks - a survey of methodological issues. Licentiate thesis. Department of Computer Science, Lund University. 130 p.

26. Markiewicz M. y Lucena C. 2001. Object oriented framework development. Crossroads 7 (4): 3-9.

27. Poulin J. 1997. Measuring Software Reuse: Principles, Practices, and Economic Models. Boston. Addison-Wesley. 195 p.

28. Solomon M. 1987. Algorithms for the vehicle routing and scheduling problems with time window constraints. Operations Research 35 (2): 254-265.

29. Soundarajan N. 2000. Documenting framework behavior. ACM Computing Surveys 32.

30. Vacic V. 2002. Vehicle Routing Problem with Time Windows. Departmente of Computer Science and Engineering. University of Bridgeport.

31. Van Gurp J. y Bosch J. 2001. Implementation and Evolution of Object-Oriented Frameworks: concepts & guidelines. Software: Practice and Experience. pp. 277-300.

32. The VRP Web. [en línea] <http://neo.lcc.uma.es/radi-aeb/WebVRP/> [consulta: julio 2005].

ANEXO 1: Formulación Matemática del VRP

Para poder apreciar la complejidad que tiene el problema de ruteo de vehículos es necesario conocer el modelo matemático que lo representa, que además es útil para diseñar soluciones que exploten su estructura.

Estudiemos el problema clásico de la sección 3.1, donde se tienen n clientes que deben ser servidos exactamente una vez por una flota de k vehículos [7].

Parámetros

a_i: demanda del cliente i

b: capacidad de los vehículos

Variables

$$y_{ik} = \begin{cases} 1 \; si\, el\, cliente\, i\, es\, visitado\, por\, el\, vehiculo\, k \\ 0 \; si\, no \end{cases}$$

$$y_k = (y_{ok}, y_{1k}, ..., y_{nk})$$

quedando el modelo así:

$$Min \quad \sum_k f(y_k)$$

$$\sum_i a_i y_{ik} \leq b \; \forall k \quad (1)$$

$$\sum_k y_{ik} = \begin{cases} k & i = 0 \\ 1 & \forall i \neq 0 \end{cases} \quad (2)$$

$$y_{ik} \in \{0,1\} \quad (3)$$

Donde (1) es la restricción de capacidad de los vehículos que no debe ser excedida por las cargas de los clientes. En la restricción (2) se asigna un vehículo a cada cliente y k vehículos a la bodega, y f(yk) es el costo del mejor ruteo (más económico) de los clientes asignados al vehículo k, que es una función no lineal, la cual podría ser reemplazada por una aproximación lineal. Sin embargo veremos otro modelo [7] que es de programación lineal entera, en cuya formulación están contenidos dos problemas bien conocidos y estudiados de optimización combinatorial, el problema del Vendedor Viajero (routing) y un problema de asignación generalizada (packing).

Variables

$$y_{ik} = \begin{cases} 1 \; si\, el\, cliente\, i\, es\, visitado\; por\, el\, vehiculo\, k \\ 0 \; si\, no \end{cases}$$

$$x_{ijk} = \begin{cases} 1 \; si\, el\, vehiculo\, k\, viaja\, directamente\, de\, i\, a\; j \\ 0 \; si\, no \end{cases}$$

$$Min \quad \sum_{ijk} c_{ij}\, x_{ijk}$$

$$\sum_{i} a_i y_{ik} \le b \;\forall k \qquad (1)$$

$$\sum_{k} y_{ik} = \begin{cases} k & i = 0 \\ 1 & \forall i \neq 0 \end{cases} \qquad (2)$$

$$y_{ik} \in \{0,1\} \qquad (3)$$

$$\left.\begin{array}{lll} \sum_{i} x_{ijk} = y_{jk} & j = 0,...,n & (4) \\ \sum_{j} x_{ijk} = y_{ik} & i = 0,...,n & (5) \\ \sum x_{ijk} \le |S| - 1 & S \subseteq \{1,...,n\}, 2 \le |S| \le n-1 & (6) \\ x_{ijk} \in \{0,1\} & & (7) \end{array}\right\} \forall k$$

Las restricciones (1) a (3) definen un problema de asignación, que garantiza que cada cliente i es asignado a un vehículo, que no se superara la capacidad y que cada ruta empieza y termina en la bodega. Si los yik son fijados para satisfacer (1) a (3) las restricciones (4) a (7) para cada vehículo definen un TSP sobre los clientes asignados a ese vehículo.