Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Trabajo de Fin de Grado

Análisis del Espacio de Búsqueda del Problema de Ruteo de Vehículos

Fitness Landscape Analysis of Vehicle Routing Problem

Óscar David Martín Cabrera

La Laguna, 6 de septiembre de 2019

D. **Belén Melián Batista**, con N.I.F. 44.311.040-E profesora Titular de Universidad adscrita al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutora

D. **Christopher Expósito-Izquierdo**, con N.I.F. 78.851.649-J investigador adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como cotutor

C E R T I F I C A (N)

Que la presente memoria titulada:

”*Análisis del Espacio de Búsqueda del Problema de Rutado de Vehículos*”

ha sido realizada bajo su dirección por D. **Óscar David Martín Cabrera**, con N.I.F. 78.646.930-V.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 6 de septiembre de 2019

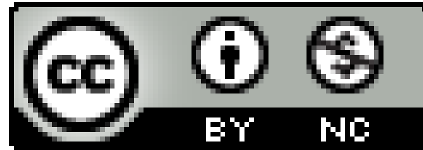
Agradecimientos

A mi familia por todo su apoyo a lo largo de la carrera, en especial a mis padres.

A mis tutores, María Belén Melián Batista, Christopher Expósito Izquierdo y Airam Expósito Márquez pues su paciencia y ayuda a lo largo del desarrollo del proyecto.

A mis compañeros por todos los recuerdos y momentos vividos durante la carrera.

Licencia



© Esta obra está bajo una licencia de Creative Commons Reconocimiento-NoComercial 4.0 Internacional.

Resumen

La optimización y resolución para problemas complejos es uno de los campos que más investigación y recursos recibe desde diferentes sectores.

El problema sobre el que se trabaja durante el desarrollo del proyecto, es el problema de ruteo de vehículos con latencia, junto a los aspectos teóricos que engloba. Se trata de lograr una resolución del problema de manera práctica y eficiente empleando varias técnicas metaheurísticas en instancias de diferentes tamaños.

Para lograr los objetivos a nivel práctico se utilizan algunas tecnologías como MongoDB, encargado del almacenamiento de las soluciones que formarán el espacio de búsqueda, CPLEX, la cual nos ayuda a comprobar que los valores de función objetivo de las soluciones alcanzadas son correctos o D3.js empleada en la representación visual del espacio de búsqueda.

Con la implementación de las técnicas VNS y LNS se busca obtener soluciones factibles al problema, al mismo tiempo que se realiza comparaciones entre las diferentes metaheurísticas a nivel práctico, contemplando de esta forma la calidad de las soluciones y el tiempo empleado por cada una de ellas en las distintas instancias. De estos resultados extraemos, por ejemplo, que la técnica VNS emplea menos tiempo para la obtención de una solución, pero suele tener menos calidad que las logradas por la LNS.

Por último, se estudia el "Fitness landscape" mediante el uso de diferentes métricas, las cuales nos indican la naturaleza del paisaje formado por un conjunto determinado de soluciones del problema, al mismo tiempo que conocemos la aptitud de cada una de ellas.

Palabras clave: VRP, LNS, VNS, Fitness landscape, optimización.

Abstract

The optimization and resolution of complex problems is one of the fields that receives more research and resources from different sectors.

The problem that is worked on during the development of the project, is the vehicle routing problem with latency as objective function, along with the theoretical aspects that encompasses. The intention is to achieve a resolution of the problem in a practical and efficient using various metaheuristic techniques in instances of different sizes.

To achieve the objectives at a practical level, some technologies are used, such as MongoDB, in charge of storing the solutions that will make up the search space, CPLEX, which helps us to verify that the objective function values of the solutions reached are correct or D3.js used in the visual representation of the search space.

With the implementation of the VNS and LNS techniques we seek to obtain feasible solutions to the problem, at the same time that comparisons are made between the different metaheuristics at a practical level, contemplating in this way the quality of the solutions and the time used by each one of them in the different instances. From these results we extract, for example, that the VNS technique takes less time to obtain a solution, but usually has less quality than those achieved by the LNS.

Finally, the "Fitness landscape" is studied through the use of different metrics, which indicate the nature of the landscape formed by a specific set of solutions to the problem, at the same time that we know the suitability of each of them.

Keywords: VRP, LNS, VNS, Fitness landscape, optimization.

Índice general

1. Introducción 1
   1.1. Contexto . . . 1
   1.2. Objetivos . . . 2
   1.3. Motivación . . . 2
   1.4. Estructura de la memoria . . . 3

2. Estado del arte 4
   2.1. Problema de ruteo de vehículos . . . 4
   2.2. Metaheurísticas . . . 5
      2.2.1. Búsqueda por entornos variables (VNS, Variable Neighbourhood Search) 6
      2.2.2. General Variable Neighborhood Search, GVNS . . . 8
      2.2.3. Búsqueda por entornos grandes (LNS, Large Neighborhood Search) . 9
   2.3. Fitness Landscape . . . 11
   2.4. Tecnologías empleadas . . . 12
      2.4.1. Java . . . 12
      2.4.2. MongoDB . . . 13
      2.4.3. ILOG CPLEX . . . 13
      2.4.4. D3.js . . . 13

3. Problema 15
   3.1. Descripción del problema . . . 15

4. Propuesta de solución 18
   4.1. Representación del problema . . . 18
   4.2. Soluciones . . . 19
      4.2.1. SequentialBuilder . . . 19
      4.2.2. RandomizedBuilder . . . 20
      4.2.3. RandomizedRCLBuilder . . . 21
   4.3. Evaluadores . . . 22
      4.3.1. Movimientos . . . 23
   4.4. Aplicadores . . . 24
   4.5. Optimización . . . 25
      4.5.1. VNS . . . 25
      4.5.2. LNS . . . 26
   4.6. Fitness Landscape . . . 27
      4.6.1. Controlador de la base de datos . . . 27
      4.6.2. Explorador del Fitness Landscape . . . 29
      4.6.3. Métricas . . . 29

4.7. Representación gráfica 31

5. Resultados computacionales **33**

5.1. VNS y LNS 33

5.2. Evaluadores 35

5.2.1. Análisis del landscape 35

6. Conclusiones y líneas futuras **37**

7. Summary and Conclusions **38**

8. Presupuesto **39**

Índice de Figuras

3.1. Representación inicial de un VRP 15
3.2. Posible solución de un VRP 16

4.1. Diagrama del SequentialBuilder 20
4.2. Diagrama del RandomizedBuilder 21
4.3. Diagrama del RandomizedRCLBuilder 22
4.4. Jerarquía de clases 23
4.5. Funcionamiento de la clase DBControl 27
4.6. Imagen general de grafo generado con el movimiento de intercambio 32
4.7. Información extra en el grafo 32

Índice de Tablas

5.1. Configuraciones VNS 33
5.2. Configuraciones LNS 33
5.3. Resultados computacionales 34
5.4. Análisis del espacio de búsqueda para intercambio 36
5.5. Análisis del espacio de búsqueda para mover después 36
5.6. Análisis del espacio de búsqueda para mover antes 36

8.1. Presupuesto 39

Capítulo 1

Introducción

1.1. Contexto

En los problemas de ruteo de vehículos en el contexto de la algoritmia, estamos ante la necesidad de llevar recursos desde un punto, normalmente denominado como depósito, desde el que parten y finalizan sus rutas los vehículos, hasta otros puntos destinos que se encuentran dispersos geográficamente y con una necesidad conocida. Cumpliendo con todas las restricciones que nos propone el problema, buscamos como objetivo principal minimizar el coste o el tiempo empleado en transportar dichos recursos.

A medida que los Problemas de Ruteo de Vehículos (VRP, por sus siglas en inglés, Vehicle Routing Problem) escalan, es imposible determinar la solución óptima en la cual tenemos una satisfacción completa por parte de los puntos destino, manteniendo el mínimo coste para hacerlo. Para las instancias que superan los 50 clientes, no existe un método exacto para obtener la solución óptima [11]. Con instancias más grandes no es posible solucionar dichos problemas en tiempo polinomial, esto indica que estamos ante un problema NP-duro. Por lo tanto necesitamos encontrar una forma de obtener eficientemente una solución de alta calidad, en la que sigamos satisfaciendo a todos los clientes y cumpliendo con las restricciones del problema, aunque no podamos asegurar que sea esa la latencia mínima para lograrlo. También tenemos que contemplar que una formulación para estos problemas puede incluir un amplio número de variables y diversos parámetros lo cual dificulta aún más la búsqueda de la solución óptima.

Con las razones comentadas hasta aquí, se estudian y proponen metaheurísticas para mejorar y analizar los diversos algoritmos usados en este proyecto para la resolución del VRP. Las metaheurísticas son estrategias y técnicas para mejorar o diseñar modelos heurísticos obteniendo un alto rendimiento en ellos, ayudando incluso a la creación de algoritmos híbridos combinando diferentes campos de investigación. Normalmente este tipo de estudios y metaheurísticas se aplican en diversos problemas de optimización y con propósito general para obtener soluciones de calidad.

La resolución de estos problemas tiene muchas aplicaciones en el mundo, principalmente en el transporte de mercancías, aunque afecta a muchos sectores, como por ejemplo la recogida de residuos de manera periódica [8] o la logística humanitaria [19]. Dada la diversidad de problemas, también sufre variaciones dependiendo de dónde se aplique. Por ejemplo como leemos en [19] el equilibrio entre la calidad de la solución y el tiempo empleado en encontrar dicha solución, cuando se trata de responder a una catástrofe es crítico. Sin embargo, en la recogida de residuos se prioriza la demanda de los clientes por encima del resto de variables.

Por tanto, para las compañías que disponen de una flota de vehículos que se desplazan habitualmente, tienen una gran dependencia de la productividad, rendimiento y seguridad de los transportes realizados por sus vehículos, así como el tiempo empleado en encontrar las rutas óptimas para cumplir sus envíos. Sus objetivos deben pasar por incrementar la eficiencia, gestionando el rendimiento y cumpliendo los objetivos gubernamentales de las zonas en las que operen.

1.2. Objetivos

Durante el desarrollo del proyecto se han cumplido los objetivos listados a continuación:

- Aprender de uso de librería de optimización.
- Implementar el VRP.
- Generar y almacenar las soluciones que forman el espacio de búsqueda.
- Desarrollar la metaheurística de Búsqueda de Entorno Variable (VNS, por sus siglas en inglés, Variable Neighbourhood Search).
- Desarrollar la metaheurística de Búsqueda por Entornos Grandes (LNS, por sus siglas en inglés, Large Neighbourhood Search).
- Implementar las métricas sobre el espacio de búsqueda.
- Visualizar el espacio de búsqueda.
- Localizar las soluciones dentro del espacio de búsqueda.

1.3. Motivación

Tal y como se ha expuesto en apartados anteriores, los problemas de ruteo de vehículos afectan a diversos sectores, proporcionando diferentes enfoques del uso de las metaheurísticas y las modificaciones necesarias para resolver problemas reales, de tal forma que una mejora significativa en el tiempo de obtención y calidad de las soluciones ayudaría a mejorar los servicios que se ofrecen a la población.

Las empresas que disponen de un departamento de transporte o asociado al movimiento de recursos, tienen un gran interés en la resolución y estudio de estos problemas. Sería muy costoso realizar un estudio específico desde cero para cada uno de los problemas que nos plantean, por tanto el poder usar o crear una técnica de propósito general que se ajuste en gran medida a las necesidades de la empresa supone una gran ayuda.

Por otra parte, los conocimientos obtenidos en el estudio, análisis y desarrollo de las diferentes técnicas y algoritmos para lograr una solución al problema, pueden ser utilizados en futuras instancias para ajustar o descartar aquellos que no ofrezcan soluciones factibles o adecuadas a las necesidades planteadas. Este análisis previo puede ahorrar un tiempo valioso desde el punto de vista de los investigadores y programadores que abordarán el problema.

1.4. Estructura de la memoria

El capítulo 2 recoge cual es la etapa y entorno en el que se encuentra actualmente el problema, así como las metaheurísticas y tecnologías empleadas. Profundizando un poco más en el problema, el apartado 3 describe más detalladamente en qué consiste, cuál es su naturaleza, el modelo matemático, etc. Continuamos con el capítulo 4 que contiene la propuesta de solución donde se muestra que soluciones son estudiadas, como se ha llegado hasta ellas, las ventajas y desventajas de cada una y el desarrollo de todas las técnicas necesarias. La sección 5 presenta los valores obtenidos de los algoritmos y el análisis práctico del espacio de búsqueda. Las conclusiones al trabajo y algunas de las posibles líneas de investigación futuras están recogidas en el capítulo 6. Por último, la sección 8 describe un posible presupuesto para la realización de este proyecto.

Capítulo 2

Estado del arte

En el capítulo anterior se ha introducido brevemente el problema tratado durante el desarrollo del proyecto, así como las características básicas del mismo. A continuación vamos a profundizar en los antecedentes al problema, el estado del arte, descripción de las metaheurísticas empleadas y las tecnologías utilizadas a lo largo del desarrollo del proyecto.

2.1. Problema de ruteo de vehículos

El Problema de Ruteo de Vehículos clásico fue propuesto por Dantzig and Ramser en 1959 [5]. En concreto, se presentó el primer enfoque algorítmico aplicado a la entrega de gasolina, donde se disponía de una flota de camiones, los cuales tenían una capacidad conocida y varias estaciones a las que suministrar el recurso desde una principal reduciendo el coste de este transporte. Este problema fue propuesto como una generalización del problema del viajante (TSP, por sus siglas en inglés, Travelling Salesman Problem) presentando un enfoque y características nuevas. Pasó a conocerse como problema de ruteo de vehículos capacitado (CVRP, por sus siglas en inglés, Capacitated Vehicle Routing Problem). Asimismo, el TSP es un caso concreto de VRP donde el vehículo (en este caso el viajante) no necesita regresar al depósito hasta visitar todos los clientes. Se puede también asumir que el VRP es un problema perteneciente al conjunto de los NP-Hard puesto que el TSP lo es. Más tarde en 1964 Clarke and Wright presentaron su algoritmo de ahorro (CWS, por sus siglas en inglés, Clarke and Wright Savings) como una solución al CVRP, la cual se convertiría en la heurística más aplicada para resolverlo [13]. Tal y como expone Gilbert Laporte en su artículo [7], el VRP y sus variantes han sido objeto de investigación durante muchos años, donde han intervenido decenas de autores creando o mejorando metaheurísticas usadas en su resolución.

Existen muchas variantes del VRP en la literatura. A continuación se citan las más conocidas:

- Problema de Ruteo de Vehículos con restricción de capacidad (CVRP).
- Problema de Ruteo de Vehículos con ventanas de tiempo (VRPTW, por sus siglas en inglés, Vehicle Routing Problem with Time Windows). En esta variante las entregas de suministros tienen que ser satisfechas en un tiempo determinado.
- Problema de Ruteo de Vehículos con múltiples depósitos (MDVRP, por sus siglas en inglés, Multiple Depot Vehicle Routing Problem). Para esta variante del problema

los vehículos tienen dos o más depósitos en los que pueden terminar o comenzar su ruta.

- Problema de entregas divididas con diferentes vehículos (SDVRP, por sus siglas en inglés, Split Delivery Vehicle Routing Problem). Esta variante establece que un cliente debe ser satisfecho por más de un vehículo.

Cuando los problemas presentan nuevas restricciones que no se contemplan en el modelo básico, como por ejemplo que los vehículos puedan realizar más de una ruta o se dispone de unas condiciones ambientales específicas, surgen nuevas variantes. En [3] o [20] se citan algunas variantes que no están recogidas en la lista anterior. Estas variaciones también se ven afectadas dependiendo del sector en el que se aplique ya que no será lo mismo enfocar el problema para el tratamiento y eliminación de residuos [1], donde se define el Problema de Ruteo de Vehículos Rollon-Rolloff (RR-VRP, por sus siglas en inglés, Rollon-Rolloff Vehicle Routing Problem) en el que los vehículos sólo pueden transportar una carga al mismo tiempo. Por otra parte en [4] se expone una variante nueva denominada, Problema de Ruteo de vehículo de 2 Escalones (2E-VRP, por sus siglas en inglés, Two-Echelon Vehicle Routing Problem) donde los recursos pasan por un satélite antes de ser entregado a los clientes y por tanto es necesario un enfoque diferente a la hora de abordar el problema. Con estos ejemplos podemos afirmar que a día de hoy el VRP y sus variantes siguen siendo material de investigación en diferentes campos, así como también aumentan sus aplicaciones en el mundo real.

2.2. Metaheurísticas

El objetivo de un problema de optimización combinatoria es encontrar la solución $x_{opt} \epsilon X \subseteq S$ que minimiza la función objetivo f, $min f(x) : x \epsilon X, X \subseteq S$, donde S es un conjunto grande pero finito que representa el espacio de soluciones, X es el conjunto de soluciones factibles, x es una solución factible y f el valor real de la función objetivo.

Existen dos tipos de algoritmos para abordar estos problemas, completos o aproximados. Los algoritmos completos garantizan que la solución dada es óptima para cada instancia de un problema combinatorio. Sin embargo, para los problemas de optimización combinatoria que son considerados NP-Hard, no existe un algoritmo completo que resuelva dicho problema en tiempo polinomial. Por otra parte, los algoritmos aproximados no aseguran que la solución encontrada sea la óptima global pero sí una solución de calidad en un tiempo mucho menor. De este último tipo normalmente se distinguen entre constructivos o de búsqueda local. En las últimas décadas han surgido nuevos algoritmos con el objetivo de explorar de manera eficiente el espacio de soluciones. Es aquí donde se introduce el término metaheurística. El primero que hizo referencia a este término fue Fred Glover en su artículo [6], el cual compuso dicho concepto combinando el prefijo griego, meta (‘más allá’ o ‘un nivel superior’) y la palabra heurístico (derivada del verbo heuriskein, ‘encontrar’).

Actualmente no existe una definición comúnmente aceptada del término metaheurística, aunque diversos autores en los últimos años han compartido su definición, como por ejemplo, *“Una metaheurística es formalmente definida como un proceso de generación iterativo el cual guía a una heurística subordinada al combinar inteligentemente diferentes conceptos de exploración y explotación en el espacio de búsqueda, aprendiendo estrategias usadas en estructurar la información con el objetivo de encontrar*

eficientemente soluciones cercanas al óptimo” [12]

Resumiendo las características fundamentales:

- Las metaheurísticas son estrategias que “guían” el proceso de búsqueda.
- El objetivo es explorar el espacio de búsqueda eficientemente para encontrar soluciones cerca del óptimo.
- Los algoritmos metaheurísticos son aproximados y normalmente no deterministas.
- Deben incorporar mecanismos para no quedarse atrapados en regiones prometedoras del espacio de búsqueda.
- No son dependientes del problema.

Por lo comentado anteriormente podemos decir que las metaheurísticas son estrategias de alto nivel para explorar espacios de búsqueda usando diferentes métodos. En este proyecto se ha trabajado con dos de ellas: búsqueda por entornos variables y búsqueda por entornos grandes.

2.2.1. Búsqueda por entornos variables (VNS, Variable Neighbourhood Search)

VNS es una metaheurística propuesta por Mladenović en 1997 [9], la cual está basada en un principio básico: Cambios sistemáticos en los vecindarios durante el periodo de búsqueda. Se define $N_k(k = 1, ..., K_{max})$ como un conjunto preseleccionado de estructuras de vecindario, y con $N_k(x)$ como el conjunto de soluciones en el $k - th$ vecindario de x. También se tienen que tener en cuenta las siguientes características:

1. Un mínimo local con respecto a una estructura no lo es necesariamente en otra.
2. Un mínimo global es un mínimo local respecto a todas las posibles estructuras de entorno.
3. Para muchos problemas, los mínimos locales están relativamente próximos entre sí.

De los puntos anteriores, se afirma que una solución óptima x_{opt} es una solución factible donde el mínimo local es alcanzado. Define $x' \epsilon X$ como un mínimo local con respecto a N_k, si no hay solución $x \epsilon N_k(x') \subset X$ tal que $f(x) < f(x')$. Después de encontrar el primer mínimo local, para problemas con varias estructuras de entornos, las metaheurísticas basadas en búsquedas tratan de resolver los puntos 1-3 por otros medios:

1. Determinísticos.
2. Estocásticos.
3. Tanto determinísticos como estocásticos.

Dentro de la metaheurística VNS existen múltiples variaciones. A continuación se nombran y describen brevemente algunas de las más usadas en la literatura:

- Búsqueda por entornos variables descendente *(Variable Neighbourhood Descent, VND)*: La característica principal de esta variación es que los cambios de estructura de entorno son realizados de forma determinística y se efectúan cuando llegamos a un mínimo local. Mediante una búsqueda *greedy* descendente se reemplaza iterativamente la solución actual por el resultado de la búsqueda local, mientras se produzca una mejora. En el Algoritmo 1 se representan los pasos principales que sigue esta variante.

Algoritmo 1: Variable Neighbourhood Descent

```
1  Inicialización;
2  Se selecciona un conjunto de estrucutras de vecindarios N_l, para l = 1, ..., l_max las
   cuales serán usadas en el descenso.;
3  Partiendo de una solución inicial x;
4  repeat
5      l = 1;
6      repeat
7          Exploración de vecindarios: Se busca el mejor vecino, x' de x(x'ϵN_k(x));
8          Aceptación: Si el punto generado es mejor que el actual, intercambia (x ← x')
           y continúa con la ejecución usando N_1(l ← 1); En otro caso aumenta
           l(l ← l + 1);
9      until l = max;
10 until condición de parada;
11 return x;
```

La solución final x debe ser un mínimo local con respecto a todas k_{max} estructuras de entorno utilizadas en el algoritmo, lo que implica que esta solución tiene más posibilidades de ser un óptimo global que si solo se utiliza una única estructura de vecindario. En el algoritmo 1, las estructuras de entorno se recorren de manera secuencial, aunque se puede desarrollar una estrategia anidada para llevar a cabo esta exploración.

- Búsqueda de entornos de variable reducida *(Reduced Variable Neighbourhood Search, RVNS)*: Este método se obtiene seleccionando soluciones aleatorias de $N_k(x)$. El RVNS es útil en instancias grades para las cuales la búsqueda local es costosa. Además, como condición de parada, generalmente se utiliza el número de iteraciones que ha pasado desde una mejora a otra, aunque también puede ser utilizado el tiempo de CPU (t_{max}). En el Algoritmo 2 se representan los pasos principales que sigue esta variante.

- Búsqueda en Entornos Variables Básico *(Basic Variable Neighbourhood Search, BVNS)*: En este caso se combinan cambios determinísticos y estocásticos de estructuras de entorno. Los cambios determinísticos serán representados por una búsqueda local, la cual consiste en elegir una solución inicial x, encontrando una dirección de descenso para x (dentro del vecindario $N(x)$) y moviéndose al mínimo de $f(x)$ dentro de $N(x)$ en esa dirección. Si existe un descenso posible la heurística continúa, en caso contrario, ésta se detendrá. En la búsqueda se pueden utilizar diferentes métodos para lograr esa mejora aunque los más comunes son: el que primero mejore (*first improve*) o el que más mejore (*best improve*). Como condicio-

Algoritmo 2: Reduced Variable Neighbourhood Search

```
1  Inicialización;
2  Se selecciona un conjunto de estrucutras de vecindarios N_k, para k = 1, ..., k_max;
3  Partiendo de una solución inicial x;
4  repeat
5      k = 1;
6      repeat
7          Fase de agitación: Genera un punto x' de manera aleatoria procedente del
           vecindario k^th de x(x'ϵN_k(x));
8          Aceptación: Si el punto generado es mejor que el actual, se cambia (x ← x') y
           continúa con la ejecución utilizando N_1(k ← 1); En otro caso incrementa el
           valor de k(k ← k + 1);
9      until k = max;
10 until condición de parada;
11 return x;
```

nes de parada se suele utilizar el tiempo máximo de CPU, un número máximo de iteraciones o las iteraciones entre dos mejoras.

Algoritmo 3: Basic Variable Neighbourhood Search

```
1  Inicialización;
2  Se selecciona un conjunto de estrucutras de vecindarios N_k, para k = 1, ..., k_max;
3  Partiebdi de una solución inicial x;
4  repeat
5      k = 1;
6      repeat
7          Fase de agitación: Obtiene un punto x' de manera aleatoria procedente del
           vecindario k^th de x(x'ϵN_k(x));
8          Búsqueda local: Aplica algún método de búsqueda local con x' como solución
           inicial; El resultado de la búsqueda se almacena en la variable x'';
9          Aceptación: Si el punto generado es mejor que el actual, lo remplaza por
           (x ← x') y continúa la ejecución con N_1(k ← 1); En otro caso incrementa el
           valor de k(k ← k + 1);
10     until k = max;
11 until condición de parada;
12 return x;
```

2.2.2. General Variable Neighborhood Search, GVNS

Al igual que las variantes descritas anteriormente, el GVNS parte desde una solución factible del problema que se esté tratando, mediante una fase de agitación se obtiene un punto aleatorio para el vecindario correspondiente. Es en la fase de búsqueda donde se encuentra la mayor diferencia con respecto a las otras variantes, pues en este caso la búsqueda se realiza por medio del VND 1. Finalmente si el resultado de la búsqueda es mejor que la solución original, se toma como nuevo punto de partida y se repite el proceso

para ese entorno, en caso contrario se cambia al siguiente entorno. Si el algoritmo explora el último de los entornos disponibles y no encuentra mejor, se detiene la ejecución y se devuelve la mejor solución encontrada.

Con la utilización del VND en la fase de búsqueda se espera mejorar la eficiencia y calidad de las soluciones con respecto a las otras variantes del VNS.

Algoritmo 4: General VNS

```
1 GVNS(x, l_max, k_max, t_max)
2 repeat
3     k ← 1;
4     repeat
5         Fase de agitación: Obtiene un punto x' de manera aleatoria procedente del
          vecindario k^th de x(x'ϵN_k(x));
6         Búsqueda local: x'' ← VND(x', l_max);
7         Aceptación: Si el punto generado es mejor que el actual, intercambia
          (x ← x'') y continúa con la ejecución usando el primer entorno, en otro caso,
          se selecciona el siguiente entorno;
8     until k = k_max;
9 until t > t_max;
```

En la literatura se han propuesto diversas formas de extender la VNS dotándolo de nuevas características enfocadas a problemas más específicos o con condiciones no contempladas en otras variantes. Algunas de estas propuestas son el VNS con Descomposición (*Variable Neighbourhood Decomposition Search, VNDS*), VNS Sesgada (*Skewed Variable Neighbourhood Search, SVNS*) , VNS Paralela (*Parallel Variable Neighbourhood Search, PVNS*), etc.

2.2.3. Búsqueda por entornos grandes (LNS, Large Neighborhood Search)

El LNS fue propuesto por Paul Shaw en 1998 [18]. Partiendo de una solución inicial se mejora por medio de dos métodos principales que destruyen y construyen un porcentaje de la solución inicial. En la técnica de destrucción dispone de un elemento que controla la destrucción parcial de la solución en cada invocación del método. Teniendo en cuenta una instancia de 100 clientes de un problema VRP y fijando un porcentaje de destrucción del 15 % existen $C(100, 15) = \frac{100!}{(15!x85!)} = 2{,}5x10^{17}$ maneras diferentes de seleccionar los clientes. Una vez que la solución ha sido destruida parcialmente, se le pasa al método de reparación el cual, mediante el uso de algún algoritmo voraz, devolverá una nueva solución completa. Por ejemplo, se selecciona de todos los clientes disponibles, aquel elemento que su inserción provoque un menor impacto en la función objetivo y repetimos el proceso hasta que no queden elementos por insertar. Se define $N(x)$ de una solución x como el conjunto de soluciones que pueden ser alcanzadas aplicando los métodos de destrucción y reparación respectivamente.

En el pseudocódigo representado en el Algoritmo 4, línea 2 contiene la variable x^b, la cual almacenará la mejor solución hallada hasta el momento, que inicialmente será la misma que la solución inicial. En la línea 4, la variable x^d contiene el resultado del método de destrucción, al cual se le pasa la solución x. Si es la primera pasada del bucle,

Algoritmo 5: Large Neighborhood Search

1 Recibe $x \leftarrow$ como una solución factible del problema;
2 $x^b \leftarrow x$;
3 **repeat**
4 $\quad x^d \leftarrow d(x)$;
5 $\quad x^t \leftarrow r(x^d)$;
6 $\quad$ si acepta(x^t, x) entonces $x \leftarrow x^t$;
7 $\quad$ si $c(x^t) < c(x^b)$ entonces $x^b \leftarrow x^t$;
8 **until** *Condición de parada*;
9 **return** x^b;

x corresponde a la solución inicial. Se define x^t al resultado de la reconstrucción una vez le pasa x^d. Se comprueba que la solución temporal x^t es una solución factible para el problema acorde a los criterios de aceptación. En la línea 7, verifica si la solución temporal ha mejorado la mejor solución encontrada hasta el momento, si se cumple dicha condición, la mejor solución será ahora x^t. El algoritmo repetirá estos pasos hasta una determinada condición de parada, por ejemplo, el tiempo máximo de CPU o un número determinado de iteraciones.

Una parte crítica a la hora de implementar el LNS es la selección del grado de destrucción que se aplicará a la solución: escogiendo un porcentaje muy bajo, el rango de soluciones a explorar será también muy bajo y dificultará la mejora. Por otra parte, si el porcentaje de destrucción es muy grande, el método de reconstrucción puede devolver soluciones muy lejos del óptimo o tardar mucho tiempo en realizar la reparación.

Con el objetivo de lograr un mejor ajuste en los parámetros de destrucción y reparación, S Ropke y D Pisinger presentaron en 2006 una variación del LNS, la cual denominaron búsqueda adaptativa por entornos grandes (*Adaptive Large Neighborhood Search, ALNS*) [16]. En esta variante proponen varios métodos de destrucción y reparación, los cuales serán elegidos mediante un parámetro a lo largo de la ejecución. Son tres los métodos diferentes expuestos para realizar la eliminación:

- Heurística de eliminación de Shaw: Fue propuesto por P. Shaw en [17]. De manera general expone que se deben seleccionar los clientes a eliminar teniendo en cuenta la relación que existe entre ellos. Si las eliminaciones son muy diferentes entre sí, no hay garantía de que las inserciones se hagan en una posición correcta o incluso obtener la misma solución.

- Eliminación aleatoria: Se selecciona un elemento i de manera aleatoria y se elimina de la solución.

- Peor eliminación: Se busca al cliente que tiene el coste más elevado y se remueve, con la esperanza de que al insertarlo de nuevo en una posición distinta se mejore la solución.

Del mismo modo existen varios métodos para realizar la reparación:

- Heurística voraz básica: Se trata de una heurística constructiva simple en la que tratamos de insertar el elemento con menor impacto para el coste total en el valor de la función objetivo. Se repite esta selección hasta que no queden elementos fuera de la solución. Por tanto en cada iteración sólo se modifica una ruta.

- Heurística *regret*: Tratan de mejorar las heurísticas voraces básicas incorporando algún tipo de información de los siguientes elementos a insertar para cada candidato a entrar en la solución.

A cada una de las diferentes heurísticas mencionadas anteriormente, se les asigna un peso, el cual será utilizado para seleccionarlas usando el principio de selección de la ruleta (*roulette wheel selection principle*). Se debe considerar que el método de inserción se selecciona de forma independiente al de destrucción (y viceversa).

Algoritmo 6: Adaptive Large Neighborhood Search

1 Recibe $x \leftarrow$ como una solución factible del problema;
2 $x^b \leftarrow x$;
3 $p^- = (1, \ldots, 1)$; $p^+ = (1, \ldots, 1)$;
4 **repeat**
5 Escoge los métodos de destrucción y reparación $d \epsilon \Omega^-$ y $r \epsilon \Omega^+$ usando p^- y p^+ respectivamente;
6 $x^d \leftarrow d(x)$;
7 $x^t \leftarrow r(x^d)$;
8 si acepta (x^t, x) entonces $x \leftarrow x^t$;
9 si $c(x^t) < c(x^b)$ entonces $x^b \leftarrow x^t$;
10 actualiza p^- y p^+;
11 **until** *Condición de parada*;
12 **return** x^b;

La línea 5 representa la selección de los métodos de destrucción y reparación. Como $p^- \epsilon\ \mathbb{R}^{|\Omega^-|}$ y $p^+ \epsilon\ \mathbb{R}^{|\Omega^+|}$ se definen los pesos usados para la selección de los métodos de destrucción y de reparación respectivamente. Se conoce a los conjuntos contenedores de los diferentes métodos como Ω^- a los asociados con la destrucción y Ω^+ para los de reparación. Estos pesos serán actualizados en la última línea del bucle, se usarán modelos matemáticos como los expuestos en [15] o [16]. Al final de la iteración del bucle se escoge el parámetro de ajuste dependiendo de el resultado de la iteración actual.

2.3. Fitness Landscape

El concepto de Fitness Landscape fue introducido por el genetista estadounidense, Sewall Green Wright en 1932 [14], a quien sin denominarlo por ese término se le atribuye el origen del mismo. Es un concepto muy utilizado por diversos autores y estudiado en diferentes sectores como la biología, medicina o la informática.

En optimización combinatoria, el número de posibles soluciones para un determinado problema es finito, pero en la gran mayoría de ellos, explorar todo el espacio de soluciones es imposible. Por tanto, se extrae una pequeña parte de todo este conjunto para analizarlo, obteniendo de esta forma la estructura del problema con la que se debe trabajar en la búsqueda de los óptimos.

Para definir un fitness landscape de una instancia de un problema dado, se debe fijar un valor de aptitud para cada solución $s \epsilon S$. Se denomina con d a la estructura del espacio, el cual asigna para cada par de soluciones un valor, $d(\cdot, \cdot)$. Por tanto, tenemos que un fitness landscape de una instancia de un problema es una tripleta $L = (S, f, d)$, donde S

es el conjunto de soluciones, $f : S \to \mathbb{R}$ es la función fitness que asocia los valores a cada solución en S y $d : SxS \to \mathbb{R}$ es la distancia entre cada par de soluciones, cumpliendo las siguientes restricciones:

- $d(s, t) \geq 0$
- $d(s, t) = 0 \Leftrightarrow s = t$
- $d(s, t) \leq d(s, u) + d(u, t) \ \forall s, t, u \epsilon S$

A continuación se enumeran una serie de propiedades que tiene un gran impacto en las búsquedas heurísticas :

- Las diferencias del valor de aptitud entre dos puntos vecinos en el espacio.
- El número de óptimos locales.
- La distribución de los óptimos locales en el espacio de búsqueda.
- La topología de atracción en los óptimos locales.

Existen varias métricas para cuantificar estas propiedades. En la sección 4.6.3 se entra en mayor detalle para aquellas que utiliza el proyecto para clasificar los diferentes landscape.

2.4. Tecnologías empleadas

A lo largo del desarrollo del proyecto se utilizan diversas tecnologías para abordar las necesidades de almacenamiento, representación e implementación propias del problema con el que se ha trabajado.

2.4.1. Java

Es un lenguaje de programación de propósito general y orientado a objetos. Fue desarrollado originalmente por James Gosling, perteneciente a la compañía Sun Microsystems en 1982. Posteriormente fue adquirida el 27 de enero de 2010 por la compañía Oracle, que es su actual distribuidora de actualizaciones y software.

Con respecto al resto de lenguajes, Java [1] tiene una particularidad, se trata de un lenguaje compilado e interpretado. Cuando se compila un programa en Java se obtiene como resultado un programa en un lenguaje intermedio denominado bytecode, el cual es posteriormente interpretado. Para interpretar los programas en bytecode es necesaria la herramienta JRE (Java Runtime Environment), del mismo modo que se necesita la JDK (Java Development Kit) para compilar los programas escritos en este lenguaje. Gracias a esta característica los programas pueden ser ejecutados y desarrollados en cualquier sistema operativo y con cualquier procesador.

Por otra parte, el proyecto utiliza una serie de librerías facilitadas y desarrolladas por lo tutores. Estas librerías están desarrolladas en Java y presentan muchas funciones y herramientas para trabajar en problemas de optimización combinatoria, desde la representación de soluciones, hasta evaluadores, constructores y muchas otras funcionalidades que facilitan el trabajo. Añadiendo así otro punto a favor para la elección este lenguaje en lugar de cualquier otro.

[1] https://www.java.com/es/

2.4.2. MongoDB

Un espacio de búsqueda en el contexto de la algoritmia está representado por todas las posibles soluciones que resuelven el problema. Dado que se trabaja con un problema NP-Hard, tal y como se ha mencionado anteriormente, es imposible contemplar y calcular todas estas soluciones, pero para que el estudio del espacio de búsqueda sea representativo, es necesario un gran número de soluciones para cada entorno. Para abordar esta necesidad se utiliza una base de datos que permita almacenar una gran cantidad de soluciones con sus respectivas características y valores.

MongoDB [2] es una base de datos NoSQL, multiplataforma y orientada a documentos de esquema libre. Gracias a que sus datos no están sujetos a un esquema estricto, es posible definir nuestra propia forma de almacenar los datos, logrando una mejor representación de las soluciones o las relaciones existentes entre ellas. MongoDB proporciona la herramienta de las colecciones para agrupar los diversos documentos, de esta manera se pueden crear conjuntos de soluciones dependiendo de la estructura que se ha usado para generarla o si es necesario guardar algún otro dato relevante sin mezclarlo con las soluciones. Los documentos son almacenados en formato BSON (Binary JSON) el cual renunciando a un poco de capacidad otorga un aumento considerable en la velocidad de consulta, la cual es una gran característica para este tipo de problemas.

Otra de las características por las que se ha seleccionado este tipo de base de datos es la facilidad a la hora de añadir, consultar, modificar o remover los datos, y la cantidad de herramientas disponibles para realizar estas operaciones en diferentes lenguajes de programación. En este caso particular presenta muchas facilidades para configurar la forma en la que se trabaja con la base de datos y aprovechando la característica de esquema libre podemos controlar todas las operaciones desde un único controlador desarrollado en Java.

2.4.3. ILOG CPLEX

La empresa IBM ofrece este entorno y sus paquetes, varias librerías C, C++, Java, .NET y Python como herramienta para solucionar problemas de programación lineal (LP) y otros problemas de optimización. Dentro de estos paquetes tenemos Concert technology, el cual es un conjunto de bibliotecas que incluyen el modelado de recursos para permitir al programador añadir optimizadores de CPLEX[3] en las aplicaciones que está desarrollando siempre que sea en C++, Java o .NET.

2.4.4. D3.js

D3.js[4] es una librería informática para la representación de datos en navegadores web. Una vez añadida a nuestra página web tendremos acceso a una gran cantidad de funciones en JavaScript con las cuales podremos representar de una manera sencilla nuestros datos. La gran mayoría de la representación de las distintas figuras geométricas o textos que utiliza son de tipo SVG, lo que nos permite para elaboraciones más complejas, añadir zoom o escalas que nos otorga una buena representación de todos los datos. Por otra parte, los datos que queremos representar deben de encontrarse en formato JSON,

[2] https://www.mongodb.com/
[3] https://www.ibm.com/es-es/products/ilog-cplex-optimization-studio
[4] https://d3js.org/

CSV o geoJSON, aunque por supuesto, podemos desarrollar nuestras propias funciones en JavaScript para capturar los datos en cualquier otro formato.

Capítulo 3

Problema

En capítulos anteriores hemos revisado el estado del arte que rodea al problema del VRP, así como una introducción al problema y a sus diferentes variantes más comunes. En este capítulo se profundizará en la parte matemática del problema.

Se puede definir el VRP como un problema de grafos, en el cual existe un nodo con características únicas que se denomina depósito y un conjunto de nodos que comparten las mismas propiedades denominados clientes. También contine un conjunto de aristas que conectan los diversos nodos, incluido el depósito. Por tanto una representación gráfica de un VRP muy básica puede ser la que se expone en la Figura 3.1. De esta misma manera es posible representar las diferentes soluciones tal y como se observa en la Figura 3.2. Estas representaciones son ideales pues muchos de los puntos están conectados entre sí de manera directa, el número de clientes es reducido y el depósito está centralizado. Sin embargo, en problemas reales esto no ocurre. Tomando como ejemplo las empresas de transporte, aunque todos los puntos sean alcanzables, el trayecto para alcanzarlo dependerá de las señales de tráfico, si es alcanzable por tierra y muchos otros factores del entorno. Teniendo esto en cuenta, asumimos que las representaciones se complican a medida que el problema escala y se acerca a los modelos reales.

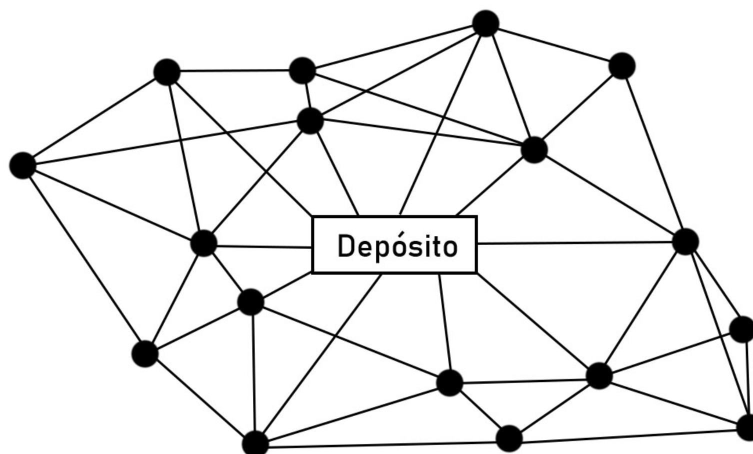


Figura 3.1: Representación inicial de un VRP

3.1. Descripción del problema

Se define un grafo $G = (N, A)$ donde $N = C \cup \{0, n+1\}$ es el conjunto de los vértices, los cuales representan los clientes contenidos en $C = \{1, \ldots, n\}$ y al depósito, el cual será

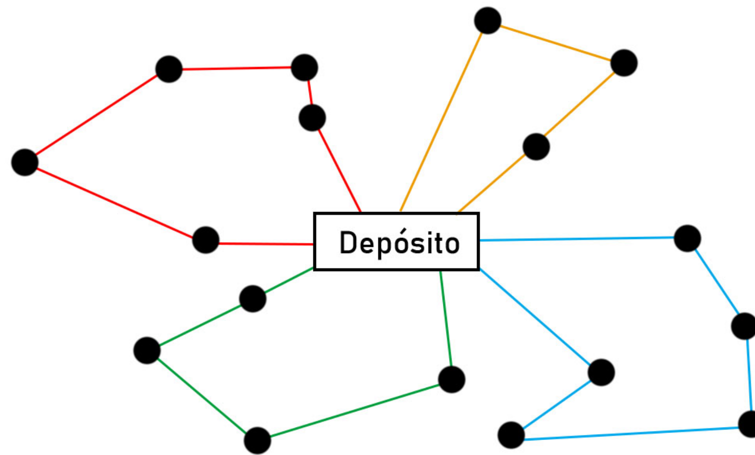


Figura 3.2: Posible solución de un VRP

representado por 0 y $n+1$ para verificar que cada vehículo comienza y acaba su ruta en el depósito. Se denomina A al conjunto de las aristas que unen los elementos de N. Cada arista (i,j), donde $i \neq j$, tiene asociada una distancia positiva c_{ij} representada en la matriz $M(c_{ij})$. Dependiendo del problema el coste podrá ser la distancia o el tiempo que se tarda en "viajar" desde el nodo i hasta el nodo j. El caso en el que M sea simétrica es posible reemplazar el conjunto A por otro conjunto E de aristas no dirigidas. Otra parte principal del VRP es el conjunto de vehículos K. Asumimos que todos tienen la misma capacidad D.

Una solución del problema debe cumplir las siguientes condiciones mínimas:

- Cada cliente en V debe ser visitado exactamente una vez por solo un vehículo.
- Todos los vehículos deben comenzar y terminar su ruta en el depósito.
- Debe satisfacer las condiciones dependientes del modelo de VRP al que nos enfrentemos.

Para algunas de las variaciones más comunes del VRP mencionadas anteriormente en 2.1, es posible contar con una o varias restricciones adicionales. Dependiendo del modelo escogido, pueden ser alguna de las que figuran a continuación:

- Restricción de capacidad: cada cliente posee asociada una demanda positiva d_i, $i > 1$. La ruta de los vehículos está condicionada por que la suma de las demandas no supere la capacidad máxima del vehículo.
- Ventanas de tiempo: Cada cliente i debe ser visitado dentro del intervalo de tiempo $[a,b]$. Los vehículos no continuarán la ruta hasta que la ventana de tiempo del siguiente punto esté habilitada; en este caso "esperarán" en el cliente que acaba de satisfacer hasta poder continuar con su ruta.

En los otros dos ejemplos se modifican las condiciones básicas del problema, donde los vehículos pueden compartir clientes o se habilita más de un depósito. Por tanto, este tipo de problemas requiere de un enfoque más específico pues se añaden restricciones que no están presentes en otras variantes del problema.

Dado que el VRP es un problema de optimización combinatoria y con todo lo mencionado anteriormente, una posible formulación matemática para el problema de ruteo de vehículos podría ser la siguiente:

Se define una variable binaria x_{ij} $(i \neq j)$ que tendrá el valor 1 si y sólo si, la arista (i,j) está incluida en el conjunto A' de una solución. En el caso de que $d_i + d_j \geq D$ los elementos i y j no pueden encontrarse en la misma ruta.

$$\begin{aligned} \text{minimizar} \quad & \sum_{\mathrm{i}\neq \mathrm{j}} \mathrm{c}'_{\mathrm{ij}} \mathrm{x}_{\mathrm{ij}} \\ \text{sujeto a} \quad & \sum_{j=0}^{n'+1} \sum_{j\neq i} x_{ij} = 1 (i = 1, \ldots, n')(1) \\ & \sum_{j=1}^{n'} x_{0j} \leq K(2) \\ & x_{ij} \in 0, 1 (i, j = 0, \ldots, n'+1; i \neq j)(3). \end{aligned}$$

En la restricción (1) se comprueba que cada cliente ha sido visitado exactamente 1 vez. Con la (2) verifica que el número de rutas ha de ser igual o menor al número máximo de vehículos y finalmente con la condición (3) establece el rango de valores que pueden tomar las distintas variables.

Esta es una formulación que solo contempla las restricciones principales de un VRP. En el caso que se quiera abordar un CVRP, deberá ser contemplada también la restricción de las capacidades máximas de los vehículos y que dicha capacidad no sea excedida en esa ruta. Del mismo modo que para un VRPTW se realizará una incorporación similar en las restricciones para limitar el espacio de tiempo en el que los vehículos pueden satisfacer a cada uno de los clientes.

Existen también variantes, como la desarrollada en este proyecto, cuya diferencia principal es la función a minimizar. En este caso se intenta minimizar la suma de todas las latencias de cada ruta, la cual viene dada por la siguiente fórmula:

$$TCT = \sum_{k=1}^{|K|} TCT_k \tag{3.1}$$

la cual es la suma de la latencia de cada ruta k, con tamaño k_r (número de clientes visitados) definida de la siguiente manera:

$$TCT_k = \sum_{i=0}^{i<k_r} d_{k_i,k_{i+1}} * (k_r - i + 1) + d_{k_r,0}, \tag{3.2}$$

donde $d_{i,j}$ es el tiempo de desplazamiento del cliente i hasta el cliente j.

Realizando el cambio en la función objetivo, el modelo matemático expuesto anteriormente de la siguiente manera:

$$\begin{aligned} \text{minimizar} \quad & \sum_{k=1}^{|K|} TCT_k, \\ \text{sujeto a} \quad & \sum_{j=0}^{n'+1} \sum_{j\neq i} x_{ij} = 1 (i = 1, \ldots, n')(1) \\ & \sum_{j=1}^{n'} x_{0j} \leq K(2) \\ & x_{ij} \in 0, 1 (i, j = 0, \ldots, n'+1; i \neq j)(3). \end{aligned}$$

Capítulo 4

Propuesta de solución

El objetivo de este capítulo es profundizar en la implementación y el desarrollo del proyecto con el fin de trasladar los conceptos teóricos expuestos en capítulos anteriores a la práctica. Para lograr este objetivo se han desarrollado una serie de clases [2] de las cuales se describen las más relevantes a continuación

4.1. Representación del problema

A lo largo de toda la implementación se utiliza, entre otras, la librería *kaizten-optimization*, facilitada por los cotutores del proyecto, la cual ofrece una gran ayuda a la hora representar las diferentes partes que componen el problema. Para usar correctamente la librería se deben extender algunas de las clases nuevas que se desarrollen, de tal forma que todas las conexiones y los objetos tengan los tipos correctos, evitando de esta manera, problemas al invocar los diferentes métodos y clases que proporciona la librería.

Cumpliendo con lo comentado en el párrafo anterior, se crea una clase que extienda de *OptimizationProblem*, la cual se denomina como *Vrp*, por las siglas en inglés del tipo de problema con el que se trabaja. Dentro de esta clase aparecen 3 variables enteras destinadas a almacenar el número de vehículos que están disponibles, los clientes que deben ser satisfechos y los clientes máximos que puede visitar cada vehículo. Cada cliente tiene información adicional, tales como las coordenadas geográficas en las que se encuentran y dependiendo del problema podrán disponer también de la demanda, el tiempo válido para las entregas y cualquier otra variable relevante para la resolución del problema. Para almacenar dichas características, y asumiendo que las condiciones descritas para estos problemas pueden ser representadas como números enteros, se crea una estructura en forma de matriz utilizando la clase *ArrayList*, de tal forma que se disponga de un array para cada cliente, donde las dos primeras posiciones almacenan las coordenadas x e y respectivamente, seguido del resto de variables si las hubiera. Una vez recopilada toda la información referente a los clientes, se genera una segunda matriz de latencia de tipo $double$. En ella se encuentran almacenadas las diferentes latencias para los nodos conectados, pudiendo acceder rápidamente a la información sin necesidad de calcularla en reiteradas ocasiones. Para este proyecto, todos los nodos se encuentran conectados y la latencia se calcula utilizando las coordenadas de los puntos por medio de la distancia euclídea. Se asume también que una unidad de distancia se recorre en una unidad de tiempo, siendo esto un entorno teórico el cual permite comprobar el correcto funcionamiento del programa para diferentes instancias del problema sin invertir mucho tiempo en crear dichas instancias.

Es necesaria una forma cómoda para suministrar los datos al programa, puesto que hacerlo estático no tendría sentido debido a que en cada llamada del programa, para una instancia diferente del problema, sería necesaria la modificación del código o pasar demasiada información por línea de comando, dificultando de esa manera un uso correcto del programa. Como resolución a este problema, se implementa la clase *VrpSupplier* la cual se encarga de leer un fichero que contiene la instancia del problema, obteniendo los atributos necesarios de cada línea y rellenando cada variable expuesta en el párrafo anterior para la clase *Vrp*. Del mismo modo, antes de devolver el objeto con el formato deseado añadirá los evaluadores correspondientes, ya sea para obtener el valor de la función objetivo o la variación que produce cada movimiento a dicha función. Con respecto a los evaluadores, se entrará en mayor detalles en la sección 4.3.

4.2. Soluciones

La librería proporciona la clase *RoutesSolution*, la cual permite representar soluciones que se basan en rutas con una estructura muy sencilla. Dispone de una serie de arrays con tamaño n (n igual al número de clientes), en los cuales se almacena la lista de predecesores, sucesores, la ruta en la que se encuentra ese cliente y si ha sido incluido ya en la solución. Existen otros arrays de características similares que no se utilizarán debido al tipo de problema que se aborda en este proyecto. También posee 3 arrays para almacenar información con respecto a las rutas. La dimensión de estos arrays serán el número de vehículos que hay disponibles para resolver el problema, en los cuales están representados el primer y último cliente que visita cada vehículo, así como el tamaño de la ruta. Finalmente en un conjunto de datos de tipo $double$ se almacenan los valores de las funciones objetivos correspondientes. Para obtener las soluciones iniciales se desarrollan 3 clases, las cuales partiendo de una representación del VRP fabrican una solución que satisface todas las restricciones. Estas clases son *SequentialBuilder*, *RandomizedBuilder* y *RandomizedRCLBuilder*. Puesto que estas clases, proveen una solución inicial, se trabaja con instancias donde el número de vehículos con su respectivas capacidades en cuanto al número de clientes que pueden satisfacer es, como mínimo, suficiente para visitarlos a todos.

4.2.1. SequentialBuilder

Tal y como nos indica su nombre, esta clase contiene un constructor secuencial. En el diagrama 4.1 se muestra que la clase recibe inicialmente un objeto de tipo *Vrp*. Después de la invocación del método *run*, el constructor cogerá el primer cliente y la primera ruta, comprobando si dicha ruta está vacía, en caso afirmativo el cliente será añadido después del depósito, en otro caso, lo añade después del último cliente que ha sido visitado por ese vehículo. Se pasa a la siguiente ruta, donde será insertado el próximo cliente disponible. El constructor, continuará con este proceso hasta que el número máximo de rutas sea alcanzado, en ese caso si aún quedan clientes fuera de la solución, se toma de nuevo la primera ruta y se repite el proceso hasta que todos los clientes pertenezcan a la solución. Finalmente evalúa la solución construida y la devuelve con tipo *RoutesSolution*.

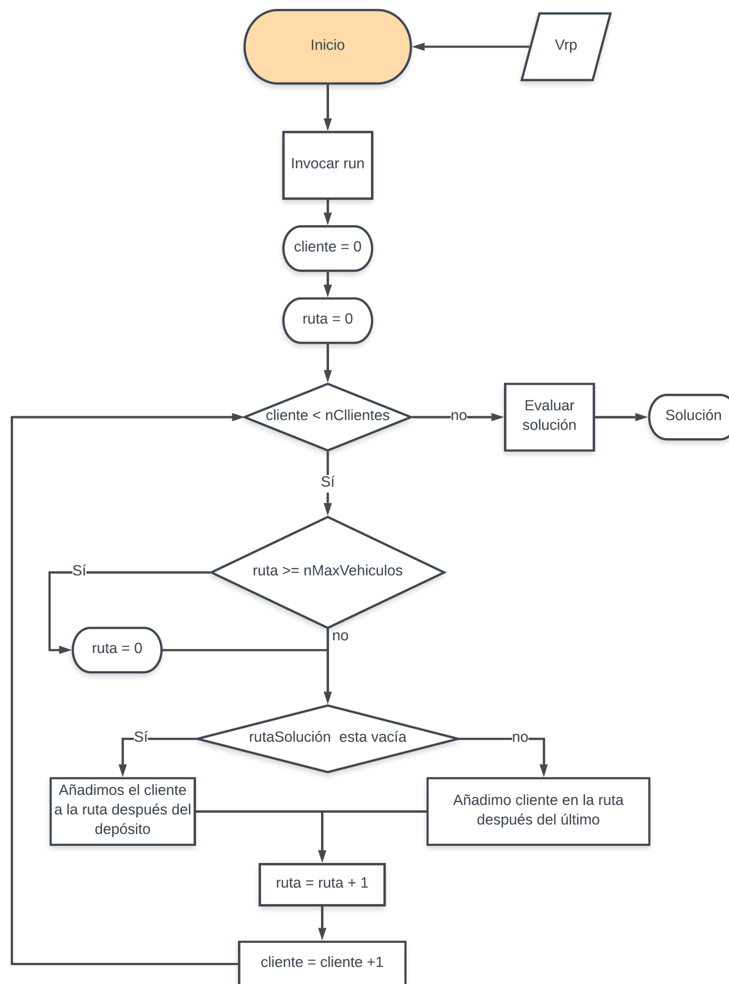


Figura 4.1: Diagrama del SequentialBuilder

4.2.2. RandomizedBuilder

Consiste en un constructor aleatorio completo, el cual también parte de una instancia de *Vrp*. En el diagrama 4.2 se representan 2 listas, una para las rutas que están disponible y otra que contiene los clientes que aún no pertenecen a la solución. Mientras queden clientes sin visitar, se selecciona uno de la lista manera aleatoria y lo remueve de dicha lista evitando que esté disponible en futuras iteraciones. Elige, nuevamente de forma aleatoria, una ruta disponible de la lista. Añade el cliente previamente seleccionado a la ruta y comprueba si con esa inserción, dicha ruta ha alcanzado el tamaño máximo permitido, en ese caso, elimina la ruta de la lista ya que no es válida para las siguientes iteraciones y repite el proceso. Por último evalúa la solución completa y la devuelve.

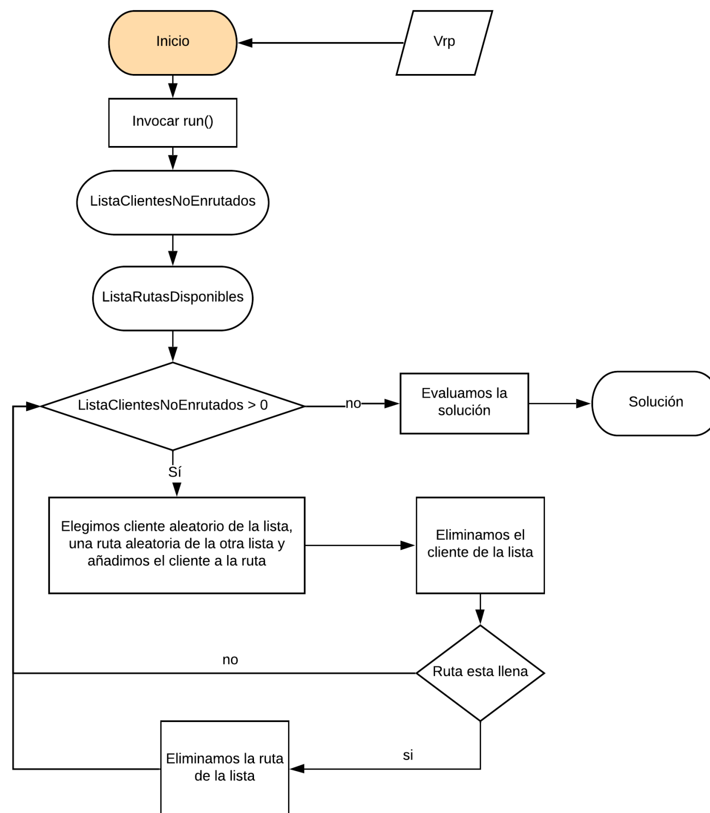


Figura 4.2: Diagrama del RandomizedBuilder

4.2.3. RandomizedRCLBuilder

A diferencia del constructor anterior, en este caso se trata de limitar la aleatoriedad a la hora de seleccionar los candidatos y se rellena la misma ruta mientras su tamaño no sea máximo. En el diagrama 4.3 se definen 2 listas: una para todos los candidatos disponibles inicialmente y otra lista que se denomina lista de candidatos restringida (Restricted Candidate List, RCL) la cual tendrá un tamaño n menor que el total de clientes. En este caso se ha trabajado con un tamaño entre 3 y 4. Se comienza rellenando la RCL con los mejores candidatos, es decir, inicialmente con los clientes que tienen menor coste de tiempo con respecto al depósito y se eliminan de la lista de disponibles. Se selecciona uno de los elementos de RCL de manera aleatoria y se le asocia al primer vehículo, después del último elemento insertado, en el caso que sea el primer cliente satisfecho por ese vehículo se inserta después del depósito. A continuación, si quedan candidatos en la primera lista, se extrae el más prometedor y se inserta en la lista restringida. Ahora se tendrá en cuenta el último cliente satisfecho para calcular cuál es el que menos latencia añade a la ruta. El constructor repite este proceso hasta que el vehículo alcanza el número máximo de clientes, en ese momento se pasa al segundo vehículo y continúa el proceso hasta obtener una solución completa. Dicha solución es evaluada y devuelta como solución factible del problema.

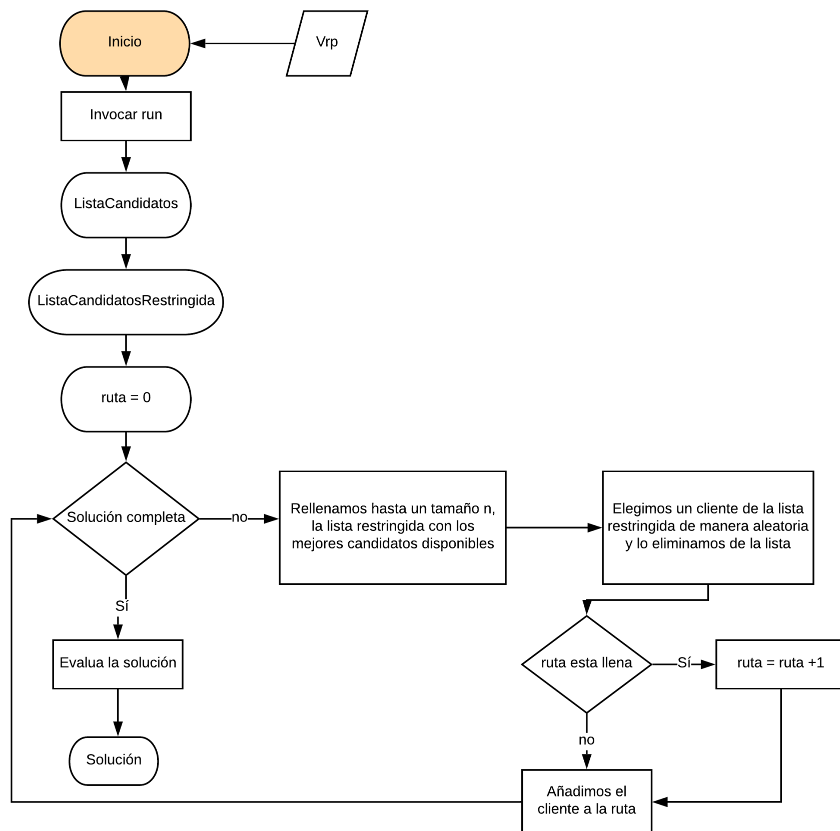


Figura 4.3: Diagrama del RandomizedRCLBuilder

4.3. Evaluadores

El trabajo de los evaluadores consiste, de manera general, en calcular el valor resultante de la función objetivo del problema para cada solución. En este caso, se han implementado la clase *EvaluatorObjectiveFunctionLatency* que extiende de la clase *EvaluatorObjectiveFunction*. De esta forma se añade un evaluador propio que más tarde se le asociará al objeto *Evaluator* el cual será el encargado de invocar cada evaluador en el momento indicado. En la parte izquierda del diagrama 4.4 están recogidas las conexiones y clases que se utilizan para cada uno de los evaluadores. Los evaluadores también son los encargados de medir cual es la desviación producida en el valor de la función objetivo de cada solución después de verse afectada por un movimiento. Por tanto clase *EvaluatorObjectiveFunctionLatency* será la encargada de replicar en código la fórmula expuesta anteriormente en la sección 3 para este problema. Teniendo en mente la representación interna de la solución descrita en el apartado 4.2 comienza su evaluación desde la primera ruta, cogiendo el primer cliente satisfecho por ese vehículo y calculando la latencia entre dicho cliente y el depósito. Una vez completada la operación, almacena el resultado en la variable denominada *objective* de tipo $double$. A continuación le suma a dicha variable la latencia entre el primer cliente de la ruta y el segundo. Luego entre el 2º y el 3º, hasta llegar al último cliente. Como último paso para este vehículo, se obtiene la latencia entre el último cliente y el depósito, sumándola de nuevo a la variable *objective*. En este punto *objective* contiene la latencia acumulada de la primera ruta, por tanto repitiendo este proceso con el resto de rutas, dicha variable contendrá el valor de la

latencia total para esa solución, cumpliendo con la fórmula propuesta para esta variación de VRP.

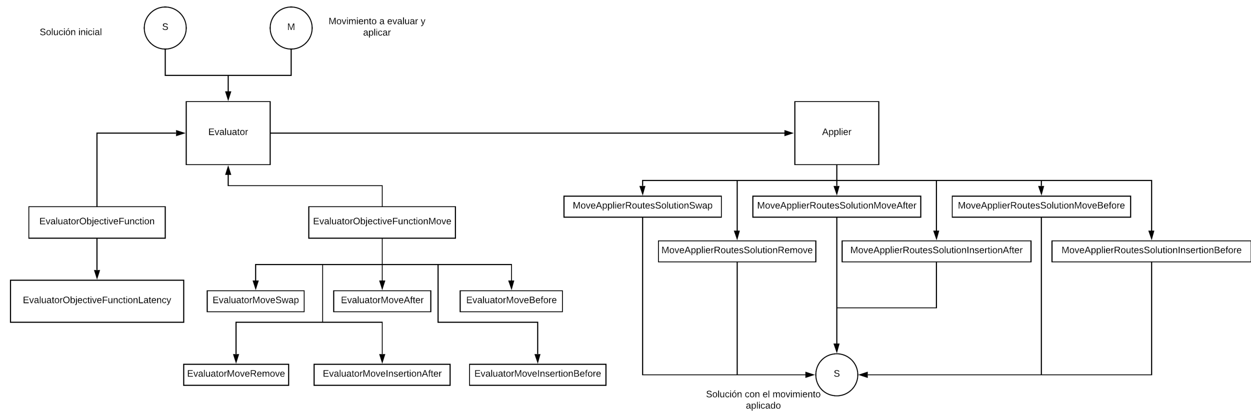


Figura 4.4: Jerarquía de clases

4.3.1. Movimientos

Se puede definir un movimiento como la alteración de alguno de los elementos que componen la solución, ya sea por el cambio de posición de uno o varios de ellos, la inclusión de aquellos que no pertenecen aún a la solución o la eliminación de aquellos que sí estaban incluidos. Desde el punto de vista de la evaluación, sería muy costoso tener que evaluar la solución completa cada vez que se ejecute un movimiento, ya que, en muchos casos el movimiento será descartado debido a que la solución resultante tiene un valor de función objetivo mayor que la original o simplemente el resultado es una solución no factible para el problema. Por todo esto, es necesario definir varios evaluadores (uno por cada movimiento que se ha desarrollado) con los que obtener la desviación producida a la función objetivo por ese movimiento, sin necesidad de evaluar la solución de nuevo.

Dado que los movimientos solo afectan a un determinado número de clientes y vehículos, se puede calcular la desviación creando una copia de las rutas que han sido modificadas. Por ejemplo, se realiza un intercambio entre el elemento 4 perteneciente a la ruta 7, con el 10 que se encuentra en la ruta 1. Calculando el valor correspondiente de las rutas antes de efectuar el cambio y después de efectuar dicho cambio, la diferencia obtenida entre estos valores es la desviación que produce ese movimiento. Por tanto, se conoce el impacto que supone realizar ese cambio en la función objetivo, ahorrando el costo de duplicar la solución y volver a evaluarla en su totalidad. Aunque todos los evaluadores de movimientos siguen el mismo principio comentado en el ejemplo, existen algunas variaciones para aquellos que pueden originar una solución infactible.

Intercambio

Se cambia la posición del elemento i con la del elemento j. En este caso la solución estará compuesta por el mismo número de clientes, por tanto, si la solución era factible, lo seguirá siendo después de realizar el movimiento ya que ninguna de las rutas ha sido modificada en el número de clientes visitados. Esto no implica que la solución sea mejor que la anterior, simplemente que sigue cumpliendo todas las restricciones del problema.

Movimiento antes y Movimiento después

Se desplaza el elemento i antes o después, del elemento j. En esta ocasión se deben tener en cuenta algunos casos concretos dependiendo del problema, puesto que este movimiento sí produce alteraciones en los tamaños de las rutas. Por ejemplo, tomando el elemento i situado en la ruta x y el j perteneciente a la ruta y, al realizar el desplazamiento se obtiene que la ruta x ha disminuido su tamaño en 1, del mismo modo que la ruta y ha aumentado esa unidad. Por tanto, si el problema tiene un valor máximo o mínimo que limita el número de clientes por ruta, cabe la posibilidad, de que el movimiento nos devuelva una solución infactible. Para este problema se ha contemplado el caso donde la ruta tiene límite máximo de clientes.

Eliminación

Este movimiento remueve el elemento i de la solución. Cada vez que se aplica este movimiento, el valor de la función objetivo disminuye, pero las soluciones resultantes son incompletas. Dependiendo de la finalidad de este movimiento, dichas soluciones pueden no ser útiles. Sin embargo, durante este proyecto se utiliza el movimiento de eliminación en combinación con los movimientos y técnicas para alcanzar nuevas soluciones. Por tanto, será necesario un análisis extra que no solo se centre en la plenitud de la solución, sino también en su uso, para determinar si son o no factibles dependiendo del entorno donde se aplique. Este movimiento también puede originar soluciones infactibles para los problemas donde no puede haber un vehículo que no atiendan a ningún cliente, aunque en este proyecto no se ha impuesto esa restricción.

Inserción antes e Inserción después

Añade el elemento i, que no se encuentra actualmente en la solución, antes o después del elemento j. Este movimiento parte de una solución incompleta, puesto que si todos los elementos se encuentran en la solución, no será capaz de encontrar uno que pueda ser insertado. Normalmente el valor de la función objetivo se verá aumentado después de aplicar la inserción, siendo mayor que el de la solución de partida. Esto es debido a que la solución vecina dispone de un elemento más que su predecesora. Otra consideración que debe tenerse en cuenta es que este cambio afecta al tamaño de las rutas, volviendo al caso en el que un límite en las rutas supondría el resultado de soluciones infactibles para determinados movimientos.

4.4. Aplicadores

En la parte derecha del diagrama 4.4 está la representación de los 6 aplicadores que están implementados, uno por cada movimiento disponible. Estas clases reciben una solución y un movimiento (ya evaluado para esa solución) con el objetivo de realizar el cambio correspondiente. El movimiento contiene almacenado una variable el valor de la desviación que produce, el cual se ha debido actualizar en la invocación del evaluador. Si el movimiento genera una solución factible, se realiza el cambio correspondiente a la solución recibida, añadiendo la desviación al valor de la función objetivo y devolviendo la solución modificada.

4.5. Optimización

Hasta este punto, se ha expuesto el desarrollo de las clases para representar los problemas, soluciones, los movimientos, los evaluadores y aplicadores. En este apartado se describirá la implementación de las técnicas mencionadas en el apartado 2.2. Puesto que la configuración de lanzamiento depende del usuario, y no siempre se lanzará para la misma instancia del problema, con los mismos vehículos, o con la misma configuración de las metaheurísticas, se solicitan una serie de variables por línea de comandos cuando se lanza la clase *Main* correspondiente. Los 2 algoritmos reciben en las primeras 4 posiciones los siguientes argumentos:

- Ruta del fichero con la instancia del problema que se desea resolver.
- Número de vehículos disponibles.
- Número máximo de clientes que puede satisfacer cada uno de los vehículos.
- Constructor para la solución inicial.

El constructor de las técnicas utiliza también los movimientos, evaluadores y aplicadores con la intención de optimizar las búsquedas locales y fases de mejora de las técnicas. Las estrucutras de entorno se encuentran disponibles en el siguiente orden:

1. Intercambio
2. Movimiento después
3. Movimiento antes
4. Eliminación
5. Insertar después
6. Insertar antes

De tal forma que el usuario pueda seleccionar los entornos que desea utilizar, añadiendo los números de las estructuras a explorar en las últimas posiciones de la lista de argumentos. Por ejemplo, una posible llamada de la técnica VNS sería: fichero.txt 7 10 2 2 1 3 la cual tratará de resolver el problema utilizando 7 vehículos, con una capacidad máxima de 10 clientes por cada uno, obteniendo la solución inicial de manera aleatoria y utilizando las estructura de entorno: movimiento después, intercambio y movimiento antes en ese orden en concreto.

4.5.1. VNS

Una vez lanzada la clase principal del *VNS* con los datos previamente descritos, se pasa a la ejecución del algoritmo. Se pueden distinguir 3 fases principales:

Fase de agitación

Con la función denominada *shake* la cual recibe por argumento el índice del vecindario con el que tiene que trabajar, se generan todos los movimientos vecinos correspondientes para la solución de partida. De todos estos movimientos se elige uno de manera aleatoria y se aplica a una copia de la solución actual, generando una solución vecina aleatoria. El cambio se aplica en una solución copia de la actual, ya que la solución generada puede ser desechada, perdiendo también la solución original sin posibilidad de recuperarla o invirtiendo demasiados recursos en deshacer el movimiento.

Fase de mejora o búsqueda local

Esta función recibe la solución generada en la fase de agitación. Mediante el uso del explorador que proporciona la librería, el método recibe la solución, el entorno que usará y el método de aceptación, para este proyecto, aquella solución que más mejore. Una vez se complete la búsqueda local se devuelve la solución resultante de esta fase.

Fase de aceptación

El objetivo de esta fase es comprobar que la solución generada por las fases anteriores es mejor que la solución actual. En el caso de que la nueva solución tenga un valor de función objetivo menor, se establece esa solución como partida y se vuelve a repetir todas las fases seleccionando la primera estructura de entorno. En caso contrario se invoca la función *shake* con la siguiente estructura. Cuando no se logre mejorar la solución y no se dispongan de más estructuras de entorno, la ejecución se detiene y devuelve la mejor solución encontrada.

4.5.2. LNS

La técnica LNS recibe dos argumentos extra, antes de las estructuras de entorno, que indican el número máximo de iteraciones y el porcentaje de destrucción que utiliza el algoritmo para este lanzamiento, además de los conocidos para la VNS.

Se crea una variable para almacenar la mejor solución encontrada, en la primera iteración dicha solución será la inicial. El método de destrucción elimina un porcentaje de clientes en una copia de esta solución. Los elementos a eliminar son seleccionados de manera aleatoria sin superar el número indicado por el porcentaje de destrucción. La solución incompleta resultante de la destrucción se le pasa al método de reparación. Este método buscará, de manera secuencial, la posición en la que la desviación producida por la inserción de ese cliente sea mínima y lo inserta.

Con la solución reconstituida, se inicia una búsqueda local a fin de encontrar aquel movimiento que produce una menor desviación para cada estructura de entorno. Una vez se localice el mejor movimiento se aplica a la solución obtenida de la reparación y se devuelve. Comprueba que el valor de la función objetivo generada por la búsqueda local es menor que el de la mejor solución encontrada. En ese caso se almacena el resultado de la búsqueda local como la mejor solución y se lanza la siguiente iteración. Una vez se alcance el número máximo de iteraciones se detiene la ejecución y devuelve la mejor solución encontrada. 0

4.6. Fitness Landscape

Dado que una parte importante del proyecto es generar y analizar los espacios de búsqueda que se obtienen aplicando los diferentes movimientos a una solución, es imprescindible almacenar las soluciones en una base de datos para lograr un volumen de información adecuado en el estudio del Fitness Landscape. Tal y como se explica en el apartado 2.4, MongoDB proporciona la estructura y velocidad perfecta para este cometido.

4.6.1. Controlador de la base de datos

Mediante el uso de los controladores y drivers que proporciona MongoDB para trabajar con Java, se desarrolla la clase *DBControl*, destinada a gestionar las operaciones de entrada y salida con la base de datos. Los objetos de esta clase reciben el nombre de la base de datos donde con la que ha de trabajar por medio del método *setDBName*. El puerto de conexión se ha dejado por defecto, que según la documentación de MongoDB es[10], 27017. Este puerto permite realizar conexiones a la base de datos por medio de los comandos de MongoDB o algún otro programa, como por ejemplo el encargado de la representación.

Dentro del controlador existen métodos de almacenamiento para añadir nuevas soluciones, almacenar la instancia de los problemas o añadir la relación de vecindad entre las soluciones. También dispone de las funciones necesarias de lectura, facilitando la recuperación de cualquiera de los objetos insertados o conjunto de ellos, como por ejemplo, obtener la lista de soluciones pertenecientes a un entorno determinado.

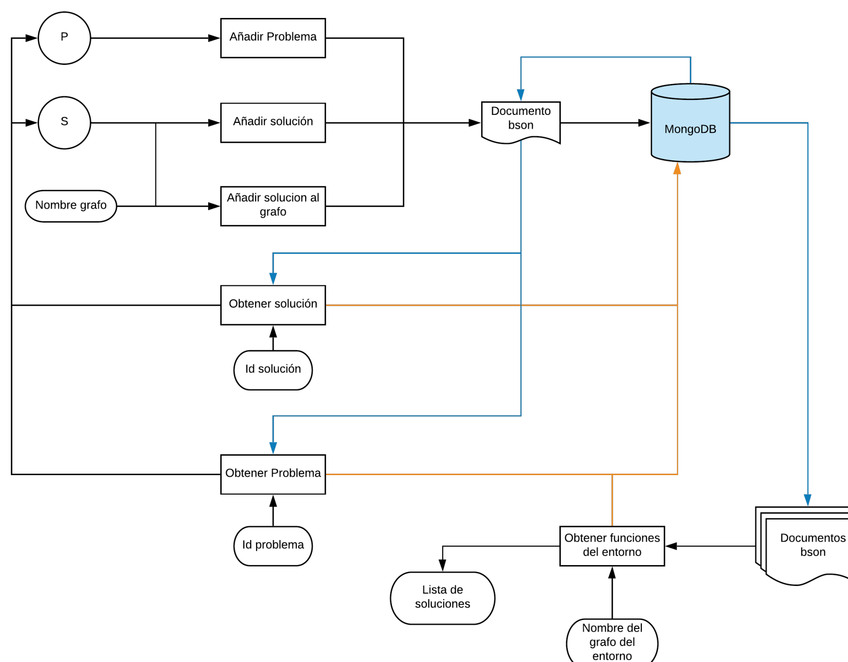


Figura 4.5: Funcionamiento de la clase DBControl

Operaciones de escritura

Tal y como se observa en la figura 4.5 la clase contiene diversos métodos para almacenar cada uno de los objetos que representan el problema. En el lado izquierdo, representado con círculos, están los tipos de datos con los que trabaja en el proyecto. Con la letra *P* se hace referencia a una instancia del problema del tipo *Vrp*, la letra *S* representa una solución del tipo *RoutesSolution*. En la parte central, se encuentran una serie de métodos cuyo fin es transformar la información los objetos anteriormente mencionados en un documento en formato BSON. Estos métodos además añaden un identificador entero que será devuelto en el momento de insertar los datos, facilitando las futuras operaciones de lectura.

Profundizando un poco más en los métodos encargados de la inserción de las soluciones, *addSolutionGraph*, crea una entrada en la colección cuyo nombre corresponde con el entorno utilizado terminado en *Graph*, por ejemplo, *moveAfterGraph* que contendrá la arista de la solución que esta siendo explorada y su predecesora. Este mismo método crea otra entrada en la colección denominada *solution* con la representación de la solución actual. De esta forma se va generando el grafo al mismo tiempo que almacena la representación de la solución.

Operaciones de comprobación

Las operaciones de escritura pueden ocasionar duplicidad, pues durante la exploración se puede generar una solución que ya se encuentre en la base de datos, lo que indicaría que la solución que está siendo explorada es su vecina. Para evitar que se inserten más de una vez cada solución y se cree el fitness landscape correctamente, se han creado varios métodos para comprobar que los diferentes objetos no han sido añadidos previamente.

Para este problema se definió que una solución es igual que otra si las listas de predecesores coinciden, pues indica que todos los elementos están distribuidos exactamente de la misma manera. Por tanto, si se realiza una consulta a MongoDB con dicha lista en la cual se obtiene un resultado distinto de *null*, significará que esa solución ya está incluida en la base de datos. Este es el propósito de los métodos *exist*. Si se da el caso que la solución o problema se encuentra en la base de datos, el método devolverá el identificador correspondiente a esa solución o problema. En caso contrario se devuelve un -1. Esto varía un poco en el caso de las aristas, ya que se devuelve un $booleano$ con el resultado de la comprobación.

Operaciones de lectura

Cuando se realiza una consulta a la base de datos, esta devuelve un fichero con el resultado codificado en formato BSON, el cual deberá ser convertido al tipo de datos correspondiente para poder ser utilizados por el resto de clases del proyecto. Para cumplir con esa tarea se crean los métodos *get*.

En el diagrama 4.5 hay varios caminos resaltados en colores, azul aparecen los resultados de las diferentes consultas a la base de datos y en naranja las consultas que realiza los métodos. Tomando como ejemplo el método para obtener una solución, este solicita el identificador de la solución y realiza la consulta para ese identificador. Una vez la base de datos ha procesado la consulta devuelve el resultado al método. Finalmente, realiza las conversiones de tipo necesarias y devuelva un objeto en formato *RoutesSolution*. De forma similar trabajan los método para obtener el problema y el conjunto de soluciones

de un entorno, salvo que este último devuelve una lista de *RoutesSolution*.

4.6.2. Explorador del Fitness Landscape

Esta clase será la encargada de ir generando las soluciones del entorno al mismo tiempo que las va almacenado en la base de datos de manera automática. Antes de lanzar el método *init* encargado inicializar todas las variables necesarias para la exploración, será necesario pasarle el nombre de la base de datos en la que el controlador ha de guardar el resultado de la exploración.

Una vez indicada la base de datos e inicializada todas las variables, se invoca al método principal *explorer*, el cual recibe 3 parámetros:

- Solución inicial desde la cual empezará la exploración.
- El entorno usado para generar el espacio de búsqueda. Sigue el mismo formato que el expuesto en la sección 4.5.
- Número de soluciones a generar.

Una vez lanzado el método, añadirá la solución original desde la que parte e irá explorando el espacio de búsqueda hasta que alcance el número máximo de soluciones. Dicha exploración consiste en ir generando secuencialmente las soluciones vecina de la solución inicial y almacenando estas soluciones en la base de datos, con sus respectivas aristas, hasta que no queden más soluciones vecinas que generar desde esa solución. En ese punto se toma como solución de partida la primera vecina generada y se realiza de nuevo la exploración hasta alcanzar el tamaño deseado para el espacio de búsqueda. Para insertar las soluciones utiliza el método *addSolutionGraph* expuesto en 4.6.1.

4.6.3. Métricas

En esta sección se describe las diferentes métricas utilizadas para analizar la naturaleza del espacio de búsqueda generado con las herramienta anterior. Se ha desarrollado una clase por cada métrica mencionada a continuación.

Distribución en el espacio de búsqueda

Para una población P, se define una distancia media como $dmm(P)$ y una la distancia media normalizada como $Dmm(P)$. La distancia de esta función corresponde al número de movimientos que se tiene que aplicar a la solución s para obtener la solución t siempre que estas soluciones sean distintas. Para lograr eso se utilizan varias funciones que determinan dicha distancia en función del movimiento utilizado para generarlas. Dentro de estas funciones se busca igualar una solución a otra mientras se contabiliza el número de movimientos necesarios para hacerlo. En lugar de trabajar sobre las soluciones directamente, se realiza una representación de las rutas almacenadas en 2 arrays, uno por cada solución y se realizan los movimientos necesarios hasta lograr la igualad en ambos arrays. Cada movimiento que se realiza en estos arrays, se incrementa la variable distancia en una unidad. Por tanto con lo expuesto anteriormente, esta métrica solo funcionará para los movimientos que incluye el proyecto, ya que para otros movimientos no será posible de calcular la distancia.

$$dmm(P) = \frac{\sum_{s \epsilon P} \sum_{t \epsilon P, t \neq s} dist(s,t)}{|P| \cdot (|P| - 1)} \quad (4.1)$$

$$Dmm = \frac{dmm(P)}{diam(S)} \quad (4.2)$$

El diámetro de una población P de S es la distancia máxima entre los elementos de P.

$$diam(P) = max_{s,t \epsilon P}\ dist(s,t) \quad (4.3)$$

Si el resultado de $dmm(P)$ es bajo significa que las soluciones están muy cercas, lo que implica que el espacio de búsqueda está focalizado en un punto. Por contra, si las distancias son muy elevadas el espacio de búsqueda está disperso.

Entropía en el espacio de búsqueda

Una entropía débil indica una concentración de soluciones. Mientras una entropía alta implica una dispersión importante de las soluciones dentro del espacio de búsqueda. Para calcular la entropía se implementa la siguiente fórmula:

$$ent(P) = \frac{-1}{n\ log\ n} \sum_{i=i}^{n} \sum_{j=1}^{n} (\frac{n_{ij}}{cardP}\ log\ \frac{n_{ij}}{cardP}) \quad (4.4)$$

A diferencia de la métrica anterior, esta es válida para cualquier espacio de búsqueda lo único que necesita el método es el conjunto de soluciones ese espacio.

Distribución en el espacio objetivo

La amplitud $Amp(P)$ de una población arbitraria P de soluciones, viene dada por la diferencia relativa entre el mejor valor de la función objetivo de la población P y el peor.

$$Amp(P) = \frac{|P| \cdot (max_{s \epsilon P}\ f(s) - min_{s \epsilon P}\ (s))}{\sum_{s \epsilon P}\ f(s)} \quad (4.5)$$

Esta métrica tampoco depende de la manera en la que se han generado las soluciones. Al igual que la anterior se basan en el valor de la función objetivo, la cual está presente en todas las soluciones sin importar la naturaleza de las mismas.
Las métricas expuestas hasta ahora, corresponden al conjunto de métricas de distribución cuyo objetivo es el análisis de la disposición de los óptimos locales en el espacio de búsqueda. A continuación se añaden una serie de métricas correspondiente a la correlación con el objetivo de analizar la robustez del paisaje junto a la correlación entre la calidad de las soluciones y su distancia al óptimo global.

El paisaje se describe como áspero si contiene muchas soluciones óptimas locales y una baja correlación entre las soluciones vecinas. Los indicadores usados para lograr este análisis son:

Tiempo de descenso

La longitud media $Lmm(P)$ del descenso relativo a la población P se puede definir como:

$$Lmm(P) = \frac{\sum_{p \epsilon P}\ l(p)}{|P|} \quad (4.6)$$

Donde $l(p)$ es la longitud del descenso empezando en la solución $p \epsilon P$. Se define tiempo de descenso como el número de soluciones que se avanza desde una solución en concreto, por medio de sus vecinas, hasta que se encuentra con una cuyo valor de función objetivo es peor. Si los tiempos de descenso son cortos, indica que el paisaje es rugoso, en caso contrario, el paisaje será suave.

Función de autocorrelación

La función de autocorrelación $p(d)$ mide correlación de las soluciones en el espacio de búsqueda con distancia d. Se aproxima usando un ejemplo grande de pares de soluciones (n soluciones con distancia d).

$$p(d) = \frac{\sum_{s,t \epsilon SxS}, dist(s,t))}{n \cdot \sigma_f^2} \tag{4.7}$$

Donde la distancia viene dada por :

$$dist(s,t) = (f(s) - \bar{f}) \cdot (f(t) - \bar{f}) \tag{4.8}$$

Correlación de las distancias de entrenamiento

Analiza en qué medida la idoneidad de una solución se relaciona con la distancia al óptimo global.

$$r = \frac{Cov(F,D)}{\sigma_f \sigma_d} \tag{4.9}$$

$$COV(F,D) = \frac{1}{n} \sum_{i=1}^{n} (f_i - \bar{f})(d_i - \bar{d}) \tag{4.10}$$

En este caso, se vuelve a depender del tipo de movimiento que se ha empleado en la generación del espacio de búsqueda, siendo la distancia el número de movimientos que separan una solución i de la solución denominada como óptimo global para este paisaje.

Salvo en la última medida de correlación, basta con aplicar los diferentes métodos al conjunto de soluciones que componen el espacio de búsqueda para poder analizarlo. En los casos concretos vale con pasarle el indicador del movimiento, tal y como ha expuesto en secciones previas. Utilizando la librería *Math* proporcionada por Java, se ha logrado la implementación de las diferentes fórmulas matemáticas definidas anteriormente, de forma rápida y eficiente.

4.7. Representación gráfica

Como propuesta de representación gráfica para los distintos espacios de búsqueda se ha desarrollado una interfaz para navegadores utilizando la librería D3.js. En la pantalla del navegador se mostrará una ventana dividida en 2 partes. La parte principal contiene el grafo correspondiente al entorno de búsqueda indicado. Si se presiona el icono en la esquina superior izquierda se despliega un menú lateral, el cual permite manejar una serie de fuerzas que la librería utiliza en cada representación 4.6. También se dispone de información adicional en cada nodo, los cuales tendrán un color determinado según el valor de la función objetivo que tenga esa solución. La escala de colores se puede

modificar desde el menú, eligiendo el color que identificará las soluciones con mejor valor en la función objetivo y el que identificará aquellas que tengan un valor peor. Las soluciones intermedias estarán pintadas con valores intermedios de la escala formada por los dos anteriores. Se podrá conocer el valor de la función objetivo exacto, junto al identificador que posee esa solución en la base de datos y las soluciones vecinas a ese nodo si se selecciona con el puntero del ratón 4.7. Para observar correctamente el grafo, se ha añadido la opción de zoom con la rueda del ratón, de tal forma que sea posible alejar el grafo para obtener una percepción más global del mismo o acercarlo para analizar cualquier detalle que requiera un enfoque más cercano. Como complemento a la herramienta de zoom y para lograr una mejor visualización es posible desplazar el grafo si mantenemos el clic izquierdo del ratón pulsado en la ventana de representación.

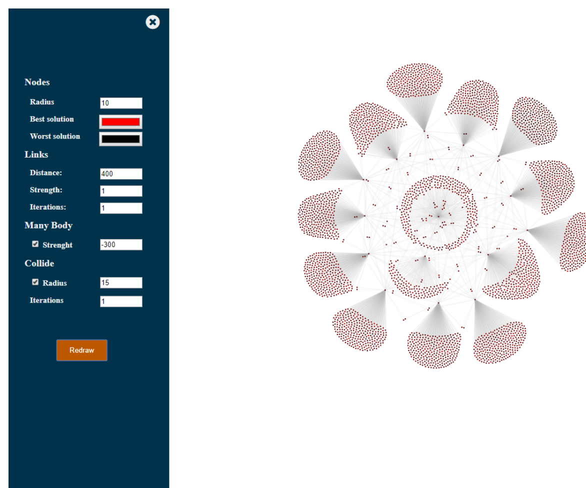


Figura 4.6: Imagen general de grafo generado con el movimiento de intercambio

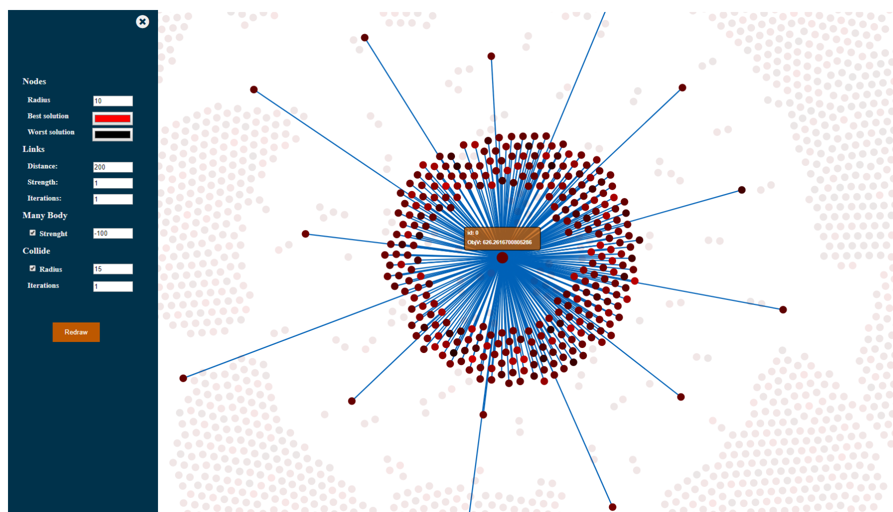


Figura 4.7: Información extra en el grafo

También se ha desarrollado un pequeño programa encargado de devolver los datos almacenados en una determinada base de datos, la cual se indicará por parámetros, un fichero en formato JSON que contenga los nodos que representan cada una de las soluciones, junto al valor de la función objetivo y la lista de aristas que forman el grafo correspondiente al entorno de búsqueda, indicado también por parámetros.

Capítulo 5

Resultados computacionales

Después de la implementación correspondiente de las diferentes técnicas expuestas en capítulos anteriores, se ha pasado a verificar su correcto funcionamiento por medio de la experimentación con diferentes configuraciones.

5.1. VNS y LNS

Se comienza con la experimentación para las técnicas VNS y LNS, utilizando un conjunto de 36 instancias de problemas diferentes separadas en 3 grupos: pequeñas, medianas y grandes, donde se encuentra almacenada la información para 25, 50 y 100 clientes respectivamente. El resto de configuraciones se obtiene por línea de comando, tal y como se explica en la sección 4.5.

Nombre de configuración	**Solución inicial**	**Estructuras de entorno**	**K**
VNS_1	Aleatoria	Mover antes e intercambio	2
VNS_2	Aleatoria	Intercambio y mover después	2
VNS_3	Secuencial	Intercambio y mover después	2
VNS_4	Secuencial	Eliminación, inserción después e intercambio	3

Cuadro 5.1: Configuraciones VNS

Nombre de configuración	**Solución inicial**	**Estructuras de entorno**	**% de destrucción**	**Iteraciones**
LNS_1	Aleatoria	Eliminación e inserción después	10 %	5000
LNS_2	Aleatoria	Eliminación e inserción antes	20 %	2500
LNS_3	Secuencial	Eliminación e inserción antes	20 %	2500
LNS_4	Secuencial	Mover antes e intercambio	20 %	5000

Cuadro 5.2: Configuraciones LNS

La tabla 5.1 representa las configuraciones escogidas para las diferentes ejecuciones de la VNS. Por otra parte en la tabla 5.2 se encuentran las configuraciones para el algoritmo LNS. El número de vehículos, así como los clientes que puede satisfacer cada uno de ellos son los mismos para la experimentación de ambas técnicas.

Los resultados obtenidos se encuentran representados en la tabla 5.3 para las diferentes configuraciones y técnicas. Cada una de estas configuraciones se ha lanzado 10 veces por cada una de las instancias, siendo las columnas, el valor medio de la función objetivo, representado como *f(x)* y el tiempo medio en milisegundos, que ha tardado en devolver la solución la VNS. Se buscan soluciones para 7 vehículos, los cuales pueden

satisfacer a un determinado número de clientes. Tratando de acercar la experimentación a un entorno real, se fija un límite máximo de clientes que cada vehículo puede satisfacer, pues en caso de no especificarlo, se puede llegar a obtener soluciones donde un único vehículo visita todos los clientes. Estos limites han sido fijados: 8, 10 y 16, en instancias pequeñas, medianas y grandes respectivamente.

	VNS_1		**VNS_2**		**VNS_3**		**VNS_4**		**LNS_1**	**LNS_2**	**LNS_3**	**LNS_4**
Nombre Fichero	f(x)	t(s)	f(x)	t(s)	f(x)	t(s)	f(x)	t(s)	f(x)	f(x)	f(x)	f(x)
C101_25.txt	307.741	40.2	343.730	36.0	401.094	32.8	377.461	33.6	245.199	232.661	226.214	220.142
C105_25.txt	235.204	41.0	272.556	41.9	271.891	42.2	335.391	28.3	155.239	155.239	155.239	155.239
C201_25.txt	335.891	41.6	340.875	33.3	364.387	48.3	369.727	24.6	222.676	226.377	225.253	216.720
C205_25.txt	341.903	37.4	340.198	41.4	369.300	46.8	400.028	24.9	224.520	224.861	222.796	216.772
R101_25.txt	378.969	43.6	396.134	36.8	457.220	40.1	439.504	27.1	259.476	253.855	253.485	246.121
R105_25.txt	366.506	40.9	376.059	33.0	378.063	48.9	417.262	27.0	258.943	252.707	252.184	247.964
R201_25.txt	390.341	45.5	371.043	50.3	419.683	56.4	391.607	25.3	258.634	252.455	254.007	246.866
R205_25.txt	373.550	42.3	426.404	34.1	420.065	38.6	463.702	27.4	258.237	252.635	254.100	245.927
RC101_25.txt	356.032	41.6	370.679	34.4	402.793	58.8	454.244	24.6	239.239	239.148	239.197	238.908
RC105_25.txt	338.514	44.2	389.575	45.7	402.258	40.1	441.753	26.6	239.005	239.661	239.200	238.908
RC201_25.txt	368.049	35.8	346.727	45.5	415.301	39.7	402.798	26.6	239.115	239.291	239.112	238.908
RC205_25.txt	369.194	42.5	383.193	43.2	425.923	31.5	389.212	26.5	239.140	239.114	239.049	238.908
C101_50.txt	721.094	121.7	719.631	124.2	761.682	96.8	916.862	37.9	508.008	500.332	498.092	439.000
C105_50.txt	709.869	149.8	697.634	115.6	732.871	114.6	923.510	36.2	509.138	500.538	499.011	436.497
C201_50.txt	865.217	112.6	856.791	106.5	874.589	132.0	1044.168	43.8	567.131	566.729	570.273	528.406
C205_50.txt	907.814	102.5	806.983	114.9	937.047	91.3	1035.035	43.5	572.248	583.072	570.943	527.956
R101_50.txt	1083.006	116.4	939.690	105.1	979.673	109.8	1080.153	56.5	618.593	614.525	609.663	540.787
R105_50.txt	1007.474	108.0	923.053	108.2	940.151	113.1	1122.913	54.7	602.536	622.259	611.937	543.503
R201_50.txt	886.457	154.2	943.930	102.8	946.512	125.8	1141.522	49.8	610.472	621.829	612.023	539.558
R205_50.txt	952.424	126.0	964.815	109.9	965.896	123.0	1136.697	57.8	614.754	613.274	620.095	542.260
RC101_50.txt	1149.861	129.1	1058.284	117.6	1208.271	102.9	1428.768	47.8	655.993	664.934	659.146	580.981
RC105_50.txt	1116.626	125.3	1032.895	125.4	1306.687	94.6	1572.099	33.6	671.942	668.442	666.292	579.228
RC201_50.txt	1114.928	122.6	1007.653	147.1	1313.923	86.5	1425.998	45.3	674.960	657.740	658.202	578.486
RC205_50.txt	1157.098	135.9	1029.132	133.2	1247.747	97.2	1514.173	39.6	689.006	671.221	664.120	581.759
C101_100.txt	2182.626	911.5	2055.740	820.1	2116.596	255.6	2168.923	83.1	1066.298	1056.465	1058.269	1032.862
C105_100.txt	2543.047	661.8	2263.438	703.6	2077.487	240.4	2193.180	77.8	1061.455	1058.356	1062.420	1033.887
C201_100.txt	2651.010	602.0	2405.453	629.1	2241.317	188.1	2333.923	72.5	1131.298	1127.040	1130.748	1082.510
C205_100.txt	2405.537	674.2	2415.681	570.0	2272.398	189.9	2360.265	62.3	1140.520	1128.450	1122.917	1084.838
R101_100.txt	2414.829	793.5	2371.404	705.8	2426.029	371.5	2808.826	127.3	1195.294	1187.987	1195.695	1148.350
R105_100.txt	2381.297	815.8	2257.810	786.0	2441.957	378.7	2827.381	122.6	1190.141	1193.324	1192.177	1142.412
R201_100.txt	2524.840	743.0	2242.449	785.8	2362.255	431.1	2741.727	136.6	1190.902	1193.131	1201.094	1141.275
R205_100.txt	2373.892	844.1	2267.318	707.4	2508.445	313.0	2898.872	96.6	1196.899	1199.559	1194.523	1143.193
RC101_100.txt	2824.289	702.9	2595.794	703.4	2827.726	336.2	3220.244	97.6	1243.048	1256.825	1247.917	1196.643
RC105_100.txt	2991.973	651.1	2725.951	636.9	2927.828	264.7	3150.008	99.4	1249.513	1246.689	1253.341	1200.617
RC201_100.txt	2820.378	726.8	2605.825	746.6	2742.758	381.9	3088.141	133.9	1250.539	1251.393	1251.984	1204.397
RC205_100.txt	2749.640	721.3	2731.201	618.8	2772.567	321.5	3169.598	104.0	1254.405	1253.141	1254.893	1198.805
Valor medio	1297.142	301.353	1229.881	286.1	1296.122	152.344	1449.602	57.853	675.125	673.479	672.378	632.766

Cuadro 5.3: Resultados computacionales

Si se observan los resultados obtenidos para la VNS, se aprecia que estos varían entre las distintas configuraciones, pero por lo general, no demasiado. Sin embargo algunas configuraciones obtienen resultados significativamente peores, por ejemplo aquellas configuraciones que tienen una solución inicial construida de manera secuencial han obtenido peores resultados, especialmente la última configuración. El impacto de la solución inicial se puede observar de manera más directa comparando las configuraciones VNS_2 y VNS_3, pues poseen la misma configuración, salvo para la construcción de la solución de partida. Estos datos referentes a la última columna de la VNS indica que también la elección de los entornos afecta negativamente a la hora de obtener buenas soluciones. En cuanto al tiempo de exploración, las 2 primeras configuraciones presentan un tiempo de exploración aceptable, lo cual implica que las soluciones de partida ha sido mejoradas al menos unas cuantas veces antes de ser devueltas, por el contrario en las dos últimas se observa para determinadas instancias que el algoritmo no ha sido capaz de explorar demasiado, devolviendo una solución poco trabajada. Esto último se debe al factor aleatorio de la fase *shake* expuesta en el capítulo 2.2. Por tanto, como conclusión de esta experimentación se puede afirmar que existen 2 configuraciones más estables que otras, obteniendo de media mejores valores en la función objetivo.

Por otra parte, contemplando los resultados de la experimentación asociada a la LNS, se aprecian valores medios muy parejos para todas las configuraciones. Profundizando más en cada uno de los datos, las configuraciones con valores más similares son LNS_2 y LNS_3, lo que indica que la solución desde la que parte la técnica no tiene un gran impacto en los resultados obtenidos. De las primeras configuraciones se puede extraer, que aunque el número de iteraciones sea menor, el algoritmo puede obtener mejores resultados aumentando el porcentaje de destrucción. Para finalizar, la última configuración combina una construcción de la solución secuencial, con un nivel alto de destrucción para un gran número de iteraciones, pero la mayor diferencia la presenta en la fase de mejora, la cual realiza una búsqueda local distinta a las anteriores, logrando mejores resultados.

Procediendo a la comparación entre los distintos algoritmos, se observa una gran diferencia en los valores obtenidos. En el caso de la LNS el tiempo invertido para lograr estas soluciones es superior al del otro algoritmo, no se ha reflejado en la tabla ya que depende de la máquina en la que se ha lanzado el programa. Sin embargo se puede afirmar que el número de iteraciones realizadas por la LNS, generalmente, será mayor que las realizadas por la VNS empleando por tanto más tiempo del que figura en 5.3. Esta diferencia, es consecuencia de las naturaleza de los algoritmos, ya que la VNS termina la ejecución cuando agota todas las estructuras sin lograr una mejoría en el valor de la función objetivo. Por contra, la LNS no termina hasta que llega al número máximo de las iteraciones, aunque se de el caso el que no mejora, el algoritmo agotará todas las iteraciones disponibles tratando de alcanzar esa mejora. Tal y como se refleja en las tablas, una buena configuración de entorno proporcionará mejores resultados en ambos algoritmos. Por tanto, se concluye que la LNS invierte más tiempo, pero logra soluciones significativamente mejores que las obtenidas con la VNS para este problema.

5.2. Evaluadores

Una parte importante de la experimentación se ha basado en verificar que todos los evaluadores desarrollados funcionan correctamente devolviendo los valores esperados para cada evaluación. Para lograr este objetivo se ha utilizado CPLEX y una serie de clases proporcionadas por los tutores.

Para comprobar el correcto funcionamiento de los evaluadores, se ha pasado la misma solución a ambos evaluadores, solicitando que la evaluaran. En el caso que se obtenga el mismo valor con el modelo matemático y con el evaluador desarrollado en este proyecto significará que para esa solución funciona correctamente. Si se repite este proceso para un número elevado de soluciones (alrededor de 10^6) existe una cierta garantia de que ese evaluador funcione siempre correctamente. A lo largo del proyecto se ha trabajado con una precisión de 4 decimales siendo esta la precisión también exigida para los evaluadores.

5.2.1. Análisis del landscape

Anteriormente se exponían las diferentes métricas utilizadas para analizar los diferentes landscape. Para esta experimentación se han utilizado los movimientos de intercambio, movimientos antes y movimiento después como manera de generar las soluciones vecinas. Se han escogido estos movimientos porque son los que devuelven y trabajan con soluciones completas.

En las tablas 5.4 5.5, 5.6 y tenemos los resultados de la experimentación para varias instancias del problema y 25000 soluciones generadas para esa instancia.

Instancia	**dmm**	**Dmm**	**Ent**	**Amp**	**Lmm**	**p**	**r**
C101_25.txt	1.9694	0.4923	$4{,}1434e^{-5}$	0.3771	0.7018	-0.4999	0.2166
R101_50.txt	1.9031	0.4757	$4{,}0127e^{-5}$	0.1616	0.6828	-0.4999	0.2084
RC101_100.txt	1.6448	0.4112	$3{,}9999e^{-5}$	0.0992	0.7525	-0.5000	0.0107

Cuadro 5.4: Análisis del espacio de búsqueda para intercambio

Los espacios de búsqueda generados con este movimiento presentan una distancia normalizada entre las soluciones y una entropía muy baja. Analizando ahora las métricas relacionadas con la correlación, se observa que los paseos no llegan de media a una unidad y el valor de la correlación es muy bajo. Con todos estos datos, se puede afirmar que el lanscape generado por el movimiento de intercambio para este problema es rugoso y con poca dispersión entre sus soluciones. Cabe destacar que para la última instancia la correlación entre estas soluciones es prácticamente 0 y su dispersión ha aumentado, por lo tanto se presenta un paisaje más rugoso y disperso que los anteriores.

Instancia	**dmm**	**Dmm**	**Ent**	**Amp**	**Lmm**	**p**	**r**
C101_25.txt	1.9019	0.1584	$4{,}0005e^{-5}$	0.2037	0.9296	-0.5000	0.3668
R101_50.txt	1.6520	0.0750	$4{,}0019e^{-5}$	0.1339	0.7275	-0.5000	-0.0047
RC101_100.txt	1.8325	0.0610	$3{,}9999e^{-5}$	0.0711	0.7366	-0.4999	0.0408

Cuadro 5.5: Análisis del espacio de búsqueda para mover después

Los resultados obtenidos en el espacio de búsqueda generado con el movimiento de mover después también indican un paisaje poco disperso y rugoso pues lo valores cumplen los mismos parámetros que en el caso anterior. Sin embargo se puede afirmar que dentro de la similitud de estos paisajes, este presenta una menor distancia normalizada y paseos más largos. Por tanto, se trata de un paisaje más compacto y suave que el generado con el movimiento de intercambio.

Instancia	**dmm**	**Dmm**	**Ent**	**Amp**	**Lmm**	**p**	**r**
C101_25.txt	1.9872	0.1324	$4{,}0005e^{-5}$	0.3051	0.6972	-0.5000	0.1052
R101_50.txt	1.8604	0.0664	$3{,}9999e^{-5}$	0.1490	0.6952	-0.5000	-0.0367
RC101_100.txt	2.3102	0.0550	$4{,}0005e^{-5}$	0.0926	0.7752	-0.4999	-0.0124

Cuadro 5.6: Análisis del espacio de búsqueda para mover antes

Como era de esperar, el fitness landscape generado con el movimiento de mover antes, presenta características similares al generado con el movimiento de mover después. Sin embargo, para la métrica de la distancia media se ha obtenido valores algo mayores que en los otros dos entornos, pero conserva una distancia normalizada inferior. También la longitud de los paseos contiene valores aún menores que los anteriores paisajes. En este caso también se clasifica como paisaje muy poco disperso y bastante rugoso.

Capítulo 6

Conclusiones y líneas futuras

Durante el desarrollo del proyecto se ha intentado abarcar el mayor número posibles de objetivos, pero debido a la complejidad del problema tratado quedan muchas líneas de trabajo por abarcar.

- Añadir movimiento complejos, que mezclen algunos de los ya existentes o desarrollar movimientos que involucren una mayor cantidad de elementos de cada solución, con el fin de llegar a estructuras de entorno nuevas.
- Desarrollar nuevos algoritmos para la resolución del VRP como por ejemplo, algoritmos genéticos, realizando un estudio del comportamiento y los resultados obtenidos por estos algoritmos, tal y como se ha realizado para las técnicas VNS y LNS.
- Plantear un VRP que contenga más restricciones como por ejemplo, la demanda de los clientes, para poder observar si las técnicas desarrolladas se comportan exactamente de la misma manera o se obtienen resultados distintos.
- Explorar otras formas de representación para el espacio de búsqueda que permita dibujar los datos en el espacio de una manera distinta a la obtenida con la librería D3.js.

Con el desarrollo de este proyecto, se observa que uno de los puntos críticos es el estudio necesario del tipo de problema al que nos enfrentamos, sobre todo, cuando se trata de un problema complejo como es el caso. Es imprescindible consultar aquellos artículos y autores que han realizado trabajos sobre el problema, ya sea en la variante que se va a trabajar o en otra que comparte algún tipo de similitud. De esta forma se consigue un mejor enfoque a la hora de proponer una posible solución, proporcionando también una idea básica de lo que se debe de obtener en la experimentación y ahorrando una gran parte del trabajo de ensayo y error. En cuanto a la implementación, el uso de librerías, el enfoque de un problema con una cantidad tan grande de posibles soluciones y el volumen de información con la que se trabaja añade un reto extra, forzando la adquisición y desarrollo técnicas de implementación que anteriormente no se tenía, logrando un cambio completo en la manera programar y abordar los problemas.

Capítulo 7

Summary and Conclusions

Throughout the development of this project has focused on the vehicle routing problem from various points of view, from the theoretical to the practical, through the learning and implementation of different techniques necessary to achieve a solution to the VRP with latency.

At this point of the project we can conclude that achieving a good solution for a problem, and especially when it is a complex problem, depends on many fields not only computer science, because all the information we can get regarding the problem and the possible uses in other areas, studies of other authors, etc. will greatly influence our decision making.

Another critical point is the experimentation and the way in which we verify that our theoretical ideas and everything developed meets the expectations for which they were designed.

Finally, in this development of solutions for complex problems, it is essential to use the tools offered by the different technologies or the libraries that various authors make available to programmers, as all these tools greatly facilitate our work.

Capítulo 8

Presupuesto

En este capítulo se recoge un posible presupuesto para llevar a cabo el proyecto.

Tareas	**Horas empleadas**	**Precio**
Consulta del estado actual del problema	30h	35€
Revisión de las librerías	20h	35€
Instalación y aprendizaje de tecnologías	50h	35€
Desarrollo de las técnicas en código	250h	50€
Ejecución de las pruebas	40h	40€
Creación de la documentación	15h	35€
Total	**365**	**18125€**

Cuadro 8.1: Presupuesto

Bibliografía

[1] R Aringhieri, G Carello, and D Morale. This is an author version of the contribution published on. *Transportation Science*, 2019.

[2] Óscar David Martín Cabrera. Landscape analysis of vehicle routing problem. https://github.com/Oscar-Dmc/Landscape-Analysis-Vehicle-Routing-Problem.

[3] Jose Caceres-Cruz, Pol Arias, Daniel Guimarans, Daniel Riera, and Angel A Juan. Rich vehicle routing problem: Survey. *ACM Computing Surveys (CSUR)*, 47(2):32, 2015.

[4] Teodor Gabriel Crainic, Guido Perboli, Simona Mancini, and Roberto Tadei. Two-echelon vehicle routing problem: a satellite location analysis. *Procedia-Social and Behavioral Sciences*, 2(3):5944–5955, 2010.

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

[6] Fred Glover. Future paths for integer programming and links to artificial intelligence. *Computers & operations research*, 13(5):533–549, 1986.

[7] Gilbert Laporte. Fifty years of vehicle routing. *Transportation Science*, 43(4):408–416, 2009.

[8] Alejandra Méndez, Marisa Pontin, M Zaletti, and Luis Chávez. Heuristicas para la resolución de un problema de ruteo de vehículos periódico real. In *MECOM 2005–VIII Congreso Argentino de Mecánica Computacional*, 2005.

[9] Nenad Mladenović and Pierre Hansen. Variable neighborhood search. *Computers & operations research*, 24(11):1097–1100, 1997.

[10] MongoDB. Mongodb documentation. https://docs.mongodb.com/manual/mongo/.

[11] Alfredo Olivera. Heurísticas para problemas de ruteo de vehículos. *Reportes Técnicos 04-08*, 2004.

[12] Ibrahim H Osman and Gilbert Laporte. Metaheuristics: A bibliography, 1996.

[13] Tantikorn Pichpibul and Ruengsak Kawtummachai. An improved clarke and wright savings algorithm for the capacitated vehicle routing problem. *ScienceAsia*, 38(3):307–318, 2012.

[14] Massimo Pigliucci. Sewall wright's adaptive landscapes: 1932 vs. 1988. *Biology & Philosophy*, 23(5):591–603, 2008.

[15] David Pisinger and Stefan Ropke. Large neighborhood search. In *Handbook of metaheuristics*, pages 399–419. Springer, 2010.

[16] Stefan Ropke and David Pisinger. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation science*, 40(4):455–472, 2006.

[17] Paul Shaw. A new local search algorithm providing high quality solutions to vehicle routing problems. *APES Group, Dept of Computer Science, University of Strathclyde, Glasgow, Scotland, UK*, 1997.

[18] Paul Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In *International conference on principles and practice of constraint programming*, pages 417–431. Springer, 1998.

[19] Omar Viera, Sandro Moscatelli, and Libertad Tansini Mercader. Logística humanitaria y su aplicación en uruguay. *Gerencia Tecnológica Informática*, 11(30):47–56, 2012.

[20] Sapti Wahyuningsih, Darmawan Satyananda, and Dahliatul Hasanah. Implementations of tsp-vrp variants for distribution problem. *Global Journal of Pure and Applied Mathematics*, 12(1):723–732, 2016.