El Doble Problema del Viajante con Múltiples Pilas

Tesis doctoral realizada por:

D. Gregorio Tirado Domínguez

Bajo la dirección de:

Dr. D. Ángel Felipe Ortega

Dra. Dª. M. Teresa Ortuño Sánchez

Junio 2009

Departamento de Estadística e Investigación Operativa

Facultad de Ciencias Matemáticas

Universidad Complutense de Madrid

D. Ángel Felipe Ortega y D^{a}. M. Teresa Ortuño Sánchez, Profesores Titulares de la Universidad Complutense de Madrid del Departamento de Estadística e Investigación Operativa, AUTORIZAN:

La presentación de la Tesis Doctoral titulada ***El Doble Problema del Viajante con Múltiples Pilas***, realizada por D. Gregorio Tirado Domínguez bajo nuestra dirección en el Departamento de Estadística e Investigación Operativa y que presenta para la obtención del grado de Doctor, con Mención Europea, por la Universidad Complutense de Madrid.

En Madrid, a 8 de junio de 2009

Ángel Felipe Ortega

M. Teresa Ortuño Sánchez

A mis padres y a mi hermano

Agradecimientos

Al Departamento de Estadística e Investigación Operativa de la UCM, por ofrecerme un ambiente propicio y los medios necesarios para la realización de este trabajo.

A Ángel Felipe Ortega, por su gran esfuerzo y dedicación durante los últimos seis años, por empezar a guiarme incluso antes de terminar la carrera durante mi trabajo académicamente dirigido, por continuar haciéndolo después durante los cursos de doctorado y el trabajo de investigación, y finalmente por hacer posible que este trabajo salga a la luz con sus sabios consejos y acertadas correcciones, y sobre todo con su constante empeño.

A M. Teresa Ortuño Sánchez, por todo el tiempo que me ha dedicado desde que un día aparecí en su despacho a hablar con ella de una tesis, por su total implicación y su tremendo entusiasmo, por sus impagables sugerencias e indicaciones, sin las cuales este trabajo no hubiera sido posible, y sobre todo por su infinita paciencia, que a veces me he permitido poner a prueba.

A Bego y a todos los compañeros y amigos que han hecho que la asistencia a mis primeros congresos haya sido una experiencia maravillosa, y me han apoyado y animado siempre.

A Alberto Ceselli, Roberto Cordone, Giovanni Righini y Olga Scotti, por hacerme sentir en casa desde el primer día que llegué a Crema y por preocuparse constantemente por mí y ayudarme en todo momento. Por su trabajo y dedicación durante los tres meses de estancia, durante los cuales aprendí muchísimo, y sobre todo por ser los culpables de que mi experiencia allí fuera tan extraordinaria.

A mis padres, porque me han dado todo y todo lo que soy se lo debo a ellos, y a mi hermano, que siempre ha estado a mi lado, me ha apoyado como el que más y, por qué no decirlo, fue el primero en ir a Crema a comprobar que los italianos me trataban bien.

A Ana, por ser como es y por apoyarme de forma incondicional aunque no pudiera dedicarle todo el tiempo que se merece.

A mis amigos, a todos y cada uno de ellos, por estar ahí siempre, también en los buenos momentos pero sobre todo en los malos.

Acknowledgements

To the Department of Statistics and Operations Research. They provided me a favorable working environment and the resources needed to perform this work.

To my supervisors, Ángel Felipe Ortega and M. Teresa Ortuño Sánchez. I greatly appreaciate their constant support and effort and their wise advice, that have been essential for this thesis to come out.

To Bego and all my workmates and friends who made me enjoy so much during the attendance to my first conferences and have always supported me.

To Alberto Ceselli, Roberto Cordone, Giovanni Righini and Olga Scotti. They made me feel at home in Crema since the first day and were wonderful hosts for me. They devoted a lot of time to me and worked very hard, but above all they turned my stay in Italy into an unforgettable experience.

To my parents and my brother. They have given me everything and this thesis is dedicated to them.

To Ana. She always supported me unconditionally although I could not devote to her all the time she deserved.

To my friends. They have always been at my side, also in the good times but above all in the hard ones.

Índice

Prólogo vii

Objetivos de la tesis xi

Summary *(Normativa RD 1393/2007, Art. 7.2.1,b)* xiii

1. Estado del arte 1

1.1. Problemas de Rutas 1

1.1.1. El Problema del Viajante (TSP) 2

1.1.2. Problemas de Rutas de Vehículos (VRP) 4

1.1.3. Problemas de Rutas por Arcos (ARP) 7

1.2. Técnicas de relajación 7

1.2.1. Relajación Lagrangiana 8

1.2.2. Descomposición de Dantzig-Wolfe 9

1.2.3. Planos de Corte 11

1.3. Heurísticas y Metaheurísticas 12

1.3.1. Búsqueda Local. Estructuras de entornos 13

1.3.2. Búsqueda en Entorno Variable 14

1.3.3. Búsqueda en Entornos Grandes 26

1.3.4. GRASP 27

1.3.5. Temple Simulado 28

1.3.6. Búsqueda Tabú 31
1.3.7. Otras heurísticas 34

2. Planteamiento y descripción del DTSPMS 37

2.1. Planteamiento del problema 38
2.1.1. Casos particulares 40
2.1.2. Campos de aplicabilidad del DTSPMS 41
2.2. Modelo de programación matemática 44
2.2.1. Variables del modelo 44
2.2.2. Función objetivo 45
2.2.3. Restricciones 45
2.3. Problemas relacionados 48
2.3.1. Problema del viajante con recogida y entrega de mercancías (TSPPD) 49
2.3.2. Problema del viajante con recogida y entrega de mercancías y restricciones de carga LIFO (TSPPDL) 51
2.3.3. Problema del viajante con múltiples pilas (TSPMS) 52
2.3.4. Problema de Rutas de Vehículos con Múltiples Pilas (MP-VRP) . . . 55

3. Complejidad y obtención de cotas para el DTSPMS 59

3.1. Complejidad del DTSPMS 59
3.2. Obtención de cotas inferiores 61
3.2.1. Relajación Lagrangiana 61
3.2.2. Descomposición de Dantzig-Wolfe. Generación de columnas 65
3.2.3. Planos de Corte 67

4. Tipología de soluciones en el DTSPMS. Estructuras de entornos 73

4.1. Una representación operativa para las soluciones 73
4.2. Generación de soluciones iniciales 77

4.2.1. Soluciones iniciales factibles 77
4.2.2. Soluciones iniciales α-factibles 80
4.2.3. Soluciones iniciales α-potenciales 80
4.3. Estructuras de entornos para soluciones factibles 81
4.3.1. Intercambio en Ruta 82
4.3.2. Intercambio Entre Pilas 88
4.3.3. Intercambio Dentro de una Pila 92
4.3.4. Reinserción 96
4.3.5. r-Permutación en Ruta 106
4.3.6. r-Permutación en Pila 110
4.3.7. r-Permutación Completa en Pila 115
4.3.8. Extensión a soluciones α-factibles 120
4.4. Estructuras de entornos para soluciones α-potenciales 120
4.4.1. Reinserción en Ruta 120
4.4.2. Reinserción en Pila 122

5. Tratamiento de la infactibilidad en el DTSPMS 125
5.1. Infactibilidad por Capacidad: soluciones α-factibles 126
5.1.1. Medidas de la Infactibilidad por Capacidad 127
5.1.2. Algoritmos de Reducción de Pilas 128
5.1.3. Complejidad de los Algoritmos de Reducción de Pilas 138
5.1.4. Relación entre los Algoritmos de Reducción de Pilas 141
5.2. Infactibilidad por Precedencias: soluciones potenciales 149
5.2.1. Medidas de la Infactibilidad por Precedencias 150
5.2.2. Proyección simple de soluciones potenciales 155
5.2.3. Proyección optimizada de soluciones potenciales 157
5.3. Combinación de infactibilidades: soluciones α-potenciales 162

5.3.1. Transformación de infactibilidades 162
5.3.2. α-Proyección de soluciones α-factibles 165
5.3.3. α-Proyección de soluciones α-potenciales 168
5.3.4. Función objetivo para la búsqueda exterior 171

6. Resolución del DTSPMS mediante heurísticas 175

6.1. Búsqueda en Entorno Variable 176
6.1.1. Búsqueda Descendente 176
6.1.2. Búsqueda en Entorno Variable Básica y General 177
6.1.3. Híbridación de la VNS con otras heurísticas 179
6.1.4. Algoritmo de Búsqueda en Entorno Variable Híbrida 182
6.2. Temple Simulado 184
6.3. Búsqueda Exterior 185
6.4. Algoritmo Genérico de Multiarranque con Intensificación 188

7. Resultados Computacionales 191

7.1. Conjuntos de instancias 192
7.2. Mejores resultados previos 193
7.3. Cotas inferiores para problemas de pequeño tamaño 193
7.4. Calibración de parámetros en las heurísticas 196
7.5. Búsqueda en Entorno Variable 200
7.5.1. Búsqueda con soluciones 0-factibles 201
7.5.2. Búsqueda con soluciones α-factibles 203
7.5.3. Mejores resultados obtenidos con VNS 211
7.5.4. Comparación de Estructuras de Entornos 211
7.6. Algoritmo Exterior 214
7.7. Búsqueda en Entorno Variable Híbrida Exterior 215

7.8. Comparativa con resultados previos 217

7.9. Comparativa Temple Simulado 220

8. Conclusiones y aportaciones. Investigación futura **223**

8.1. Conclusiones 223

8.2. Principales aportaciones 226

8.3. Líneas futuras de investigación 227

Conclusions *(Normativa RD 1393/2007, Art. 7.2.1,b)* **229**

Bibliografía **233**

Prólogo

Desde la introducción del problema del viajante, hace más de medio siglo, los problemas de rutas de vehículos han sido objeto de un amplio estudio en la literatura especializada, considerándose multitud de versiones con características muy distintas. Sin embargo, los constantes cambios a los que se han visto sometidas las industrias del transporte y la logística durante los últimos años, junto con los nuevos requerimientos relacionados con el aumento de complejidad de las estrategias de planificación, han propiciado la aparición de nuevos problemas de rutas de vehículos, muchos de los cuales incluyen restricciones complejas como pueden ser restricciones de precedencia y carga y aúnan elementos que antes pertenecían a campos perfectamente diferenciados. Uno de estos problemas es el Doble Problema del Viajante con Múltiples Pilas (DTSPMS, del inglés *Double Traveling Salesman Problem with Multiple Stacks*), que es un problema de rutas de vehículos con restricciones de precedencia que fue introducido en Petersen (2006) a raíz de un proyecto de colaboración de la Universidad Técnica de Dinamarca con una empresa de software informático.

Se trata de un problema que consiste en encontrar la forma óptima de atender una serie de encargos consistentes en recoger cierta mercancía en determinadas localizaciones de una región y entregarla en las correspondientes localizaciones de otra región alejada geográficamente de la primera. La carga a transportar está formada habitualmente por euro-palés de tamaño estándar, que aunque pueden contener mercancías de muy diverso tipo, tienen unas dimensiones fijas que facilitan el uso de distintos medios de transporte y permiten su almacenaje en varias filas en el contenedor del vehículo. Los vehículos son de carga trasera y por razones de seguridad los conductores no están autorizados a manipular la carga transportada en ningún caso, por lo que no es posible ningún tipo de reorganización durante todo el proceso y los primeros encargos recogidos deben ser los últimos en entregarse.

Así, es necesario decidir qué ruta seguir para realizar la recogida de mercancía en la primera región, de qué manera almacenar dicha mercancía en el contenedor del vehículo y qué ruta seguir para entregar la mercancía en la segunda región, teniendo en cuenta las restricciones de espacio y maniobrabilidad del vehículo. El objetivo final es minimizar el

coste total de la operación, que depende casi exclusivamente de la longitud de las rutas a cubrir.

El DTSPMS es una generalización del Problema del Viajante (TSP, del inglés *Traveling Salesman Problem*) y por tanto es un problema NP-duro. En realidad es un problema notablemente más difícil de resolver que el TSP debido a las restricciones de precedencia y de carga que se añaden al problema, las cuales le permiten adaptarse mejor a diversas situaciones prácticas, y no se conocen algoritmos capaces de resolver instancias de tamaño realista en un tiempo de cómputo razonable.

La presente monografía, dedicada al estudio del DTSPMS, está estructurada en ocho capítulos. Su contenido se resume brevemente a continuación.

En el Capítulo 1 se introducen los problemas de rutas de vehículos, comentándose las variantes con mayor relevancia y proponiéndose referencias adicionales. También se presentan los elementos básicos de algunas técnicas de relajación utilizadas para la obtención de cotas inferiores, como son la Relajación Lagrangiana, la Descomposición de Dantzig-Wolfe y la Generación de planos de corte. Por último se realiza una revisión de los algoritmos heurísticos y metaheurísticos más utilizados en la literatura para la resolución de problemas de optimización combinatoria (Búsqueda en Entorno Variable, Temple Simulado, Búsqueda Tabú, etc.), y en particular para la resolución de problemas de rutas de vehículos.

En el Capítulo 2 se plantea y describe con detalle el Doble Problema del Viajante con Múltiples Pilas. Se introducen algunos casos particulares interesantes y se comentan los campos de aplicabilidad del problema que tienen una mayor relevancia. También se presenta una formulación como modelo de programación matemática, se revisan algunos problemas relacionados directamente con el DTSPMS que ya han sido abordados en la literatura y se introduce una generalización del problema.

En el Capítulo 3 se estudia la complejidad del DTSPMS, que es un problema NP-duro, y a continuación se aplican varias técnicas de relajación al problema con el objetivo de obtener cotas inferiores lo más ajustadas posible. Se propone un modelo de programación matemática con un grupo nuevo de variables que permite la descomposición del problema en cinco subproblemas más sencillos a través de la Relajación Lagrangiana. También se considera la descomposición de Dantzig-Wolfe derivada de la Relajación Lagrangiana propuesta, cuyo problema maestro se resuelve mediante un procedimiento de generación de columnas. Por último se propone un algoritmo de separación específico para el DTSPMS que se utiliza para la introducción de planos de corte en la relajación del problema en la que se eliminan las restricciones de precedencia.

En el Capítulo 4 se propone una representación nueva para las soluciones factibles del DTSPMS en función de asignaciones y permutaciones, que es más adecuada que la formulación mediante variables binarias para el diseño de las estructuras de entornos y

algoritmos heurísticos, y se introducen las soluciones denominadas α-factibles y potenciales, que son soluciones que incumplen algunas restricciones del problema pero que resultarán útiles para flexibilizar y mejorar el proceso de búsqueda. Se describen varios algoritmos para generar soluciones iniciales factibles, α-factibles y α-potenciales, que pueden ser utilizados en la etapa inicial de cualquier heurística para la generación de soluciones diversas.

Se introducen también cinco nuevas estructuras de entornos, que se unen a las dos que ya existían en la literatura, para la realización de búsquedas locales sobre el espacio de soluciones factibles del problema. Dichas estructuras de entornos se describen con rigor, proporcionando una descripción matemática precisa de todas ellas, estudiando el tamaño de los entornos que se forman a partir de cada estructura y haciendo especial hincapié en la descripción de los movimientos a realizar para obtener soluciones vecinas y mantener la factibilidad, que luego serán utilizados en el diseño de heurísticas de búsqueda en entorno variable. También se comenta cómo se pueden extender estas estructuras de entornos a soluciones α-factibles y se introducen otras estructuras nuevas para soluciones potenciales.

El Capítulo 5 se dedica al tratamiento de la infactibilidad en el DTSPMS, a través del manejo de las soluciones α-factibles (que pueden exceder las restricciones de capacidad, originando la denominada Infactibilidad por Capacidad, IC) y potenciales (que pueden incumplir las restricciones de precedencia, originando la denominada Infactibilidad por Precedencias, IP). Para eliminar la infactibilidad IC sin aumentar el coste de las soluciones se proponen varios algoritmos, estrechamente relacionados entre sí, para los cuales se da una descripción detallada y se estudia su complejidad. La eliminación de la infactibilidad IP es más compleja; para ello se proponen también varios operadores de proyección, que en este caso sí pueden alterar el coste de las soluciones modificadas. Además, se introducen distintas medidas para evaluar la infactibilidad de las soluciones consideradas y poder dirigir el proceso de búsqueda de forma adecuada.

Se analiza la relación que existe entre los dos tipos de infactibilidad considerados, concluyendo que la infactibilidad IC siempre puede transformarse en infactibilidad IP pero que el recíproco no es cierto, y se estudia cómo ambas infactibilidades pueden combinarse en un proceso de búsqueda conjunto con soluciones α-potenciales. Cabe destacar la inclusión de una amplia colección de ejemplos que ilustran los nuevos conceptos introducidos en este capítulo.

En el Capítulo 6 se proponen varios algoritmos heurísticos para la resolución del DTSPMS que utilizan las herramientas introducidas en los dos capítulos anteriores, y se presenta un pseudocódigo detallado para cada uno de ellos. Los primeros algoritmos propuestos se basan en la aplicación de las estrategias de Búsqueda en Entorno Variable al DTSPMS y en la utilización de las estructuras de entornos introducidas en el Capítulo 4. Posteriormente se propone un algoritmo específico para el DTSPMS, denominado Búsqueda en Entorno Variable Híbrida, que también se basa en la aplicación de la Búsqueda

en Entorno Variable y el uso de todas las estructuras de entornos, pero incluye algunos elementos novedosos de otras heurísticas que lo hacen más eficiente.

A continuación se propone una adaptación al DTSPMS de un algoritmo de Temple Simulado en el que se utiliza la estructura de entornos de Reinserción para la realización de la búsqueda local, y después se introduce un algoritmo novedoso, denominado algoritmo Exterior, que se basa en la realización de una búsqueda local sobre el espacio de soluciones α-potenciales del problema. Para guiar el proceso de búsqueda y conseguir soluciones finales factibles se utilizan las herramientas para manejar la infactibilidad introducidas en el Capítulo 5: medidas de infactibilidad, algoritmos de reducción, operadores de proyección, etc.

El algoritmo Exterior y la Búsqueda en Entorno Variable Híbrida se combinan de forma exitosa dando lugar a un algoritmo combinado, denominado Búsqueda en Entorno Variable Híbrida Exterior, que ha ofrecido resultados altamente satisfactorios en la práctica. Por último, para finalizar el capítulo se propone un Algoritmo Genérico de Multiarranque con Intensificación que sirve de base para la utilización conjunta de distintas heurísticas, permitiendo dedicar distinta cantidad de recursos a cada una, y para la consideración de varias soluciones iniciales a partir de las cuales comenzar el proceso de búsqueda.

El Capítulo 7 se dedica a la presentación y análisis de los resultados computacionales obtenidos aplicando los algoritmos presentados en los capítulos anteriores a la resolución de instancias del DTSPMS de distinto tamaño. Se utilizan instancias de la literatura con 12, 18, 33 y 66 encargos, sobre las cuales se puede realizar una comparación directa de los resultados obtenidos con las heurísticas propuestas en este trabajo y los mejores resultados previos publicados en la literatura por otros autores, y también se generan instancias nuevas con 132 encargos para evaluar el comportamiento de los algoritmos al ser aplicados a instancias de mayor tamaño. Además, la calibración de los parámetros de las heurísticas se realiza sobre instancias generadas a tal efecto, que son diferentes de las que finalmente se van a resolver.

De los resultados presentados se concluye que la introducción de soluciones α-factibles mejora notablemente la eficiencia de los algoritmos de Búsqueda en Entorno Variable, y que éstos se comportan mejor, en general, que el algoritmo Exterior. Sin embargo, el algoritmo que muestra un mejor comportamiento global es el algoritmo que combina ambas técnicas, que es denominado Búsqueda en Entorno Variable Híbrida Exterior. Se consiguen soluciones de buena calidad con poco tiempo de cómputo, mejorando los mejores resultados publicados en la literatura por otros autores, y se encuentran nuevas mejores soluciones en varias de las instancias consideradas.

En el Capítulo 8 se destacan las conclusiones más importantes que se derivan de esta tesis, se resumen sus principales aportaciones y se comentan algunas líneas futuras de investigación, y por último se presenta una amplia bibliografía.

Objetivos de la tesis

Los principales objetivos de la tesis son los siguientes:

- Formular y describir detalladamente un problema de rutas de vehículos con restricciones adicionales de precedencia y carga, como es el Doble Problema del Viajante con Múltiples Pilas (DTSPMS), que ha sido poco tratado por el momento en la literatura especializada, y relacionarlo con otros problemas parecidos estudiados con más detalle.
- Estudiar la complejidad del problema y obtener buenas cotas inferiores para instancias del mayor tamaño posible.
- Desarrollar algoritmos heurísticos capaces de proporcionar soluciones de buena calidad para instancias de tamaño realista utilizando poco tiempo de cómputo.
- Diseñar nuevas estructuras de entornos adaptadas al DTSPMS que permitan explotar la estructura de las soluciones del problema y guiar los procesos de búsqueda local en base a distintos criterios.
- Abordar el tratamiento de la infactibilidad en el DTSPMS, ya que es un problema con un espacio de búsqueda fuertemente restringido y con soluciones difíciles de caracterizar.
- Realizar una extensa experiencia computacional con instancias de distintos tamaños para evaluar el comportamiento de los algoritmos utilizados para la resolución del DTSPMS y presentar, relacionar y comentar con detalle los resultados obtenidos.

Summary

The changing requirements of transportation and logistic companies and the growing complexity of planning strategies have recently induced the appearance of new vehicle routing problems, many of which include complex constraints as precedence or loading constraints and merge matters that used to belong to separated fields. One of these problems that have appeared during the last few years is the Double Traveling Salesman Problem with Multiple Stacks (DTSPMS), that is introduced in Petersen and Madsen (2006) as one of the main problems of a joint project with a computer software company. The classical vehicle routing problems have been very much studied during the last decades, but the DTSPMS introduces into these kind of problems some new elements that have not been treated jointly in detail in the literature, yet.

We are dealing with a routing problem in which some pickups and deliveries must be performed in two independent networks, verifying some precedence and loading constraints imposed on the vehicle. There is a single vehicle in which repacking is not allowed, but the load can be packed in several compartments that must obey the Last-In-First- Out (LIFO) principle, while there are no mutual constraints between two different compartments. From now on the compartments of the container will be referred as stacks. Two independent regions that are supposed to be very far apart are considered, and one item must be picked up in every location of the first region and delivered to the corresponding location of the second region.

In the real life application that originally motivated the problem, the items to be delivered were standardized Euro Pallets, that could be used to load different kinds of goods. The vehicles used to carry the goods were read-loaded, making it necessary to consider a LIFO policy in the distribution of items, and they had a 40-foot pallet container in which standardized Euro Pallets fit 3 by 11, suggesting the use of 3 available stacks with a maximum capacity of 11 items. The driver was not allowed to touch the goods due to security and insurance reasons, and as a consequence repacking was not permitted during the whole process. The DTSPMS instances used for testing were generated to match all these real life conditions. Apart from this original real life application, the DTSPMS can be applied to other real situations in which some items must be picked up in one region and

delivered in another region and the load cannot be rearranged, due to insurance reasons or the existence of heavy or hazardous materials that cannot be moved easily.

The DTSPMS generalizes the classical TSP, since any TSP instance can be transformed into a DTSPMS instance in which all delivery locations are placed in the same point, and thus the DTSPMS is also an NP-hard problem. In fact, the precedence constraints imposed by the stacks make it quite more difficult to solve than the TSP. The largest instances that could be solved to optimality have 20 orders, so heuristics are clearly needed to find good solutions for instances with a realistic size.

The DTSPMS combines pickup and delivery problems (Cordeau et al., 2006) with the TSP. The Traveling Salesman Problem with Pickup and Delivery (TSPPD) has been treated during the last years using heuristics (Bianchessi and Righini, 2007 and Hernández-Pérez and Salazar-González, 2004) and exact methods (Dumitrescu et al., 2008 and Hernández-Pérez and Salazar-González, 2007); imposing LIFO loading conditions a new problem called the TSPPD with LIFO loading (TSPPDL) is obtained, which presents more similarities with the DTSPMS (with only one stack). Exact approaches (Carrabs, Cerulli and Cordeau, 2007 and Cordeau et al., to appear) and heuristics based on VNS (Carrabs, Cordeau and Laporte, 2007 and Cassani and Righini, 2004) and column generation for the Multi-Vehicle version (Xu et al., 2003) have been recently proposed to solve the TSPPDL. More complex problems including different loading conditions in 2 or 3 dimensions and allowing items to be heterogeneous have also been approached in Doerner et al. (2007), Gendreau et al. (2006, 2008) and Iori et al. (2007). Finally, Petersen and Madsen (2009) propose different heuristic approaches based on Tabu Search, Simulated Annealing and Large Neighborhood Search for the DTSPMS.

Objectives of the thesis

The main objectives of the thesis are summarized next:

- Formulate and describe a vehicle routing problem with additional precedence and loading constraints, called the Double Traveling Salesman Problem with Multiple Stacks (DTSPMS), that has not been treated in detail in the literature yet, and relate it with other similar problems that have already been approached in the literature.
- Study the complexity of the problem and obtain good lower bounds for the largest possible instances.
- Develop heuristic algorithms to provide good quality solutions for real sized instances using little running time.

- Design new neighborhood structures adapted to the DTSPMS to exploit the structure of the solutions of the problem and guide the search process according to different criteria.

- Study the treatment of infeasibility on the DTSPMS, that is a problem with a very constrained feasible space and with solutions that are difficult to characterize.

- Make an extensive computational study with different sized instances to evaluate the performance of the proposed algorithms and comment with detail the results obtained.

Thesis outline

The content of this thesis, that consists of eight chapters, is summarized next.

In Chapter 1 a brief survey on vehicle routing problems, relaxation techniques (Lagrangian Relaxation, Column Generation and Cutting Planes) and heuristics and metaheuristics (Variable Neighborhood Search, Simulated Annealing, Tabu Search, etc.) is presented. In Chapter 2 the Double Traveling Salesman Problem with Multiple Stacks is fully described and formulated, and some interesting applications to real life are commented. A formulation of the DTSPMS as a mathematical programming model is presented, some related problems that have already been approached in the literature are mentioned and a generalization of the problem is introduced.

The complexity of the DTSPMS, that is NP-hard, is studied in Chapter 3, in which some relaxation techniques are applied to the problem to try to obtain tight lower bounds. A new formulation leading to a Lagrangian Decomposition into five independent subproblems is proposed and also the associated Dantzig-Wolfe decomposition is considered, but the results obtained by these approaches have not been satisfactory. A different approach based on a cutting plane procedure to eliminate the conficts between orders belonging to a pair of independent optimal pickup and delivery tours has worked better and some lower bounds have been improved.

In Chapter 4 a new representation of the solutions of the DTSPMS in terms of permutations and assignments is proposed, making it easier to define neighborhood structures and moves between solutions. Apart from feasible solutions, two new kinds of solutions are introduced: solutions with extra capacity (violating capacity constraints) and solutions with conflicts (violating precedence constraints). These are non-feasible solutions that will be useful to diversify the search process and make it more flexible. Different algorithms to generate initial varied good quality solutions are proposed, concerning feasible solutions but also solutions with extra capacity and solutions with conflicts.

Five new neighborhood structures for feasible solutions are developed and fully described, giving a detailed mathematical description of all of them and emphasizing the description of feasible moves to create neighbors. They are extended to solutions with extra capacity, and two more new structures to deal with solutions with conflicts are also proposed.

Chapter 5 is dedicated to the treatment of infeasibility on the DTSPMS, that is introduced by the use of solutions with extra capacity (IC, Infeasibility by Capacities) and solutions with conflicts (IP, Infeasibility by Precedences). Different algorithms to eliminate both kinds of infeasibility and produce feasible solutions out of non-feasible ones are developed, and also several measures to evaluate the infeasibility of a given solution and guide and control the search process are proposed.

The relationship between IC and IP infeasibilities is analyzed, concluding that IC infeasibility can always be transformed into IP infeasibility without changing the routing part of the solution but the reciprocal statement is not true. How to combine both kinds of infeasibilities into a joint search process is discussed and many examples to illustrate all new concepts are provided.

In Chapter 6 different heuristics for the DTSPMS using the tools introduced in the two previous chapters are proposed, providing a detailed pseudocode for all of them. Standard Variable Neighborhood Search (VNS) algorithms designed with the neighborhood structures introduced in Chapter 4 are presented, and a new specific algorithm (Hybridized Variable Neighborhood Search, HVNS) adding some elements from other heuristics to the basic VNS procedure is developed.

Apart from VNS algorithms, another original algorithm called Exterior Algorithm (EXT) that is based on performing a local search on solutions with extra capacity and solutions with conflicts is proposed. The tools introduced in Chapter 5 are used to guide the search process and obtain final feasible solutions: infeasibility measures, algorithms to eliminate infeasibilities, etc. HVNS and EXT are combined successfully into a joint algorithm called Exterior Hybridized Variable Neighborhood Search (EHVNS) that has provided high quality results.

Chapter 7 is dedicated to present and analyze the computational results obtained applying the algorithms proposed in the previous chapters to solve DTSPMS instances with different sizes. Some instances from the literature with 12, 18, 33 and 66 orders are used and some new ones with more orders and for calibration purposes are generated.

From the presented results it can be concluded that the introduction of solutions with extra capacity highly improves the performance of VNS algorithms, that perform better, in general, than the EXT algorithm, and also improve the best results published in the literature by other authors. However, the algorithm with the best overall performance is

the EHVNS combining both VNS and EXT techniques, with which good quality solutions for all instances are obtained using little running time and the best known solutions of several considered instances are improved.

In Chapter 8 the most important conclusions that follow from the thesis and the main contributions derived from it are presented, together with some promising future research lines that are likely to yield interesting results. Lastly, an extensive bibliography is presented.

Capítulo 1

Estado del arte

Los problemas de rutas de vehículos han sido objeto de un amplio estudio en la literatura especializada durante las últimas décadas, considerándose una gran variedad de versiones con características muy distintas. La complejidad de gran parte de estos problemas hace que no se puedan resolver instancias de tamaño realista en un tiempo de cómputo razonable. Por ello es necesaria la utilización de técnicas de relajación, que permiten abordar el problema inicial a partir de otros problemas más sencillos, y de algoritmos heurísticos, que no garantizan la obtención de soluciones óptimas pero que pueden ofrecer soluciones de buena calidad en poco tiempo de cómputo. A continuación se introducen los problemas de rutas y algunas de las técnicas más importantes utilizadas para su resolución.

En la Sección 1.1 se presenta una introducción a los problemas de rutas de vehículos, describiendo brevemente algunas de sus variantes más importantes y estudiadas en la literatura especializada. En la Sección 1.2 se repasan los elementos básicos de algunas técnicas de relajación como la Relajación Lagrangiana, la Descomposición de Dantzig-Wolfe y la generación de Planos de Corte. Por último, en la Sección 1.3 se introducen las heurísticas y metaheurísticas más utilizadas en la resolución de problemas de optimización combinatoria, y en particular en problemas de rutas de vehículos.

1.1. Problemas de Rutas

Los problemas de rutas de vehículos, en sus múltiples variantes y versiones, han sido extensamente tratados durante las últimas décadas. A continuación se presenta una breve introducción a este tipo de problemas, que se organiza de la siguiente manera: en la primera sección se introduce el Problema del Viajante, que es el primer problema de rutas que fue estudiado con detalle y constituye el origen del conjunto de problemas que se van a tratar; en la segunda sección se presentan las principales variantes de los Problemas de Rutas de

Vehículos, que surgen al incluir en los modelos diversas consideraciones prácticas de la vida real; por último, en la tercera sección se introducen los denominados Problemas de Rutas por Arcos y se proponen algunas referencias que permiten un estudio más amplio de los mismos.

1.1.1. El Problema del Viajante (TSP)

El Problema del Viajante (conocido por sus siglas en inglés TSP: *Traveling Salesman Problem*) es quizás el problema de optimización combinatoria más estudiado a lo largo de la historia, dando lugar a multitud de variantes y extensiones con infinidad de aplicaciones prácticas en la vida real. El enunciado de este problema es simple: un vendedor debe visitar todas las ciudades de un determinado territorio una y sólo una vez y volver al punto de partida, y desea encontrar una ruta que minimice la distancia total recorrida. A pesar de su aparente sencillez aún no se ha conseguido encontrar un algoritmo que sea capaz de resolverlo en general en un tiempo razonable.

El origen del problema no está del todo claro: en el siglo XIX el matemático irlandés W. R. Hamilton propone un juego (llamado el Icosaedro de Hamilton) que guarda mucha relación con el TSP, consistente en encontrar un ciclo hamiltoniano en unas determinadas condiciones, y en un libro que data de 1832 se menciona el problema y se proponen algunas rutas por Alemania y Suiza a modo de ejemplo, pero en ningún caso se realiza un tratamiento matemático del problema; el TSP en su forma general fue propuesto por primera vez en las universidades de Vienna y Harvard en 1930, destacando el trabajo de Karl Menger, que define el problema con precisión, propone un algoritmo sencillo para encontrar soluciones factibles y comprueba la no optimalidad de la heurística greedy (partiendo de la ciudad inicial, visitar en cada momento la ciudad más cercana a la actual).

En los años 50 y 60 el problema fue ganando popularidad muy rápido dentro de los círculos científicos de la época, destacando el trabajo de George Dantzig, Delbert Ray Fulkerson y Selmer M. Johnson (1954), que modelizaron el TSP como un problema de programación lineal entera y desarrollaron un algoritmo de planos de corte para su resolución con el cual fueron capaces de resolver una instancia con 49 ciudades. Investigadores de diversos campos científicos (matemáticas, física, química, informática) comenzaron a estudiar el problema, y en 1972 Richard M. Karp demostró que el TSP es NP-duro, explicando la gran dificultad computacional con la que investigadores anteriores se habían encontrado. En los años 70 y 80 se realizaron grandes progresos y se consiguieron resolver instancias de hasta 2392 ciudades, utilizando planos de corte y algoritmos de ramificación y acotación.

En los 90 Applegate, Bixby, Chvátal, y Cook desarrollaron el software *Concorde*, que aún se utiliza en la actualidad, y en 1991 Gerhard Reinelt publicó la TSPLIB, una colección de instancias de distintos tamaños que ha sido muy utilizada por investigadores posterio-

res para comparar la eficiencia de distintos algoritmos. En la última década se han ido resolviendo problemas de dimensiones cada vez mayores; en 2006 David Applegate, Robert Bixby, Vaš ek Chvátal, William Cook, Daniel Espinoza y Marcos Goycoolea (Applegate et al., 2006) resolvieron con el Concorde una instancia de 85900 vértices de forma exacta.

Lawler et al. (1985) y Jünger et al. (1995) son dos referencias básicas sobre el Problema del Viajante. Multitud de referencias adicionales e información de todo tipo sobre el TSP pueden encontrarse en el sitio web http://www.tsp.gatech.edu/index.html.

El Problema del Viajante tiene infinidad de aplicaciones a situaciones de la vida real pertenecientes a muy diversos ámbitos, incluso restringiéndose a su forma más elemental: planificación logística, reparto de correo o mercancía de cualquier tipo, manufactura de microchips y circuitos integrados, secuenciación del genoma, y un largo etcétera. Sin embargo, a partir de la formulación inicial del TSP, innumerables variantes y extensiones han ido apareciendo en la literatura para representar otras muchas situaciones reales interesantes.

Añadiendo la posibilidad de recoger y entregar mercancías a distintos clientes dentro de una misma ruta aparece el Problema del Viajante con Entrega y Recogida de mercancías (conocido por sus siglas en inglés TSPPD: *Traveling Salesman Problem with Pickup and Delivery*), en el que se tienen que satisfacer los encargos de una serie de clientes, los cuales requieren la recogida de cierta mercancía en un determinado lugar de origen (*pickup*) y la posterior entrega en el correspondiente destino (*delivery*). El vehículo de que se dispone tiene una capacidad de carga limitada, siendo el objetivo del problema encontrar un ciclo hamiltoniano de coste mínimo que cumpla con todas las recogidas y entregas programadas respetando las restricciones de precedencia entre las localizaciones origen y destino y no excediendo la capacidad máxima del vehículo. El TSPPD fue tratado por primera vez por Stein (1978), que propuso un algoritmo heurístico para el caso de un vehículo sin límite de capacidad. Otras referencias más recientes son Gendreau, Laporte y Vigo (1999), donde se propone una metaheurística de búsqueda tabú y una heurística basada en el algoritmo de Christofides (1976) para el TSP, y Baldacci et al. (2003), que proponen un algoritmo exacto de tipo Branch & Bound.

Es habitual también la inclusión de restricciones de precedencia y/o de carga, presentes frecuentemente en situaciones de la vida real, en algunos de los problemas mencionados anteriormente, obteniéndose nuevos problemas que están empezando a recibir gran atención en la literatura especializada. El hecho de que muchos de los vehículos utilizados para el transporte de mercancías sean de carga trasera provoca que en algunos problemas de rutas con entrega y recogida de mercancía se deba imponer un orden LIFO (Last-In-First-Out, el primer encargo en recogerse debe ser el último en entregarse), dando lugar al problema denominado TSPPD con orden LIFO (conocido por sus siglas en inglés TSPPDL: *TSPPD with LIFO Loading*). El TSPPDL ha sido abordado con algoritmos heurísticos basados en la búsqueda en entorno variable en Carrabs et al. (2007b), Cassani (2004) y Cassani y Righini

(2004), y con métodos exactos en Carrabs et al. (2007a) y Cordeau et al. (2009). Además, Xu et al. (2003) abordaron la versión con múltiples vehículos (Multi-vehicle TSPPDL) con un método de generación de columnas. Otros problemas más complejos en los que los contenedores de los vehículos se representan en espacios de 2 ó 3 dimensiones, añadiendo restricciones de carga notablemente más complicadas, también han sido considerados en Doerner et al. (2007), Gendreau et al. (2006, 2008), Iori et al. (2007) y Petersen y Hansen (2009).

1.1.2. Problemas de Rutas de Vehículos (VRP)

La utilización de flotas con varios vehículos da lugar a un extenso conjunto de problemas que se denominan Problemas de Rutas de Vehículos (conocidos por sus siglas en inglés VRP: *Vehicle Routing Problems*). Todos estos problemas tienen como objetivo el diseño de varias rutas, una para cada vehículo de la flota, con coste total mínimo, verificando una serie de condiciones que pueden ser muy variadas.

En la versión básica del VRP, denominada Problema de Rutas de Vehículos con Capacidades (y conocido por sus siglas en inglés CVRP: *Capacitated Vehicle Routing Problem*), se dispone de una flota homogénea de m vehículos con cierta capacidad máxima para repartir una determinada cantidad de mercancía desde un depósito central a una serie de clientes; el objetivo del problema es diseñar m rutas que partan del depósito, visiten varios clientes satisfaciendo su demanda y vuelvan al depósito, sin exceder la capacidad máxima de los vehículos y repartiendo toda la mercancía exigida con un coste total mínimo. Este problema fue introducido por Dantzig y Ramser en 1959 y experimentó un rápido desarrollo durante las siguientes décadas (Christofides et al., 1979, Christofides, 1985 y Golden y Assad, 1988). Con el fin de ajustarse mejor a los problemas de la vida real, a partir de la versión básica surgen numerosas variantes con restricciones adicionales. Algunas de las más importantes se resumen a continuación:

- MDVRP: Problema de Rutas de Vehículos con Múltiples Depósitos (en inglés *Multi Depot VRP*). Es un VRP en el cual hay varios depósitos desde los cuales pueden comenzar y terminar las rutas. Puede haber algunos vehículos asociados a algunos de los depósitos, y en ocasiones se permite que las rutas comiencen en un depósito y terminen en otro distinto. Fue introducido por Kulkarni y Bhave (1985), que propusieron varias formulaciones para el VRP y el MDVRP.
- PVRP: Problema de Rutas de Vehículos Periódico (en inglés *Periodic VRP*). En el VRP clásico el horizonte temporal es de un solo día, esto es, todos los vehículos parten al mismo tiempo y entregan sus mercancías en un solo trayecto visitando una sola vez a cada cliente y sin regresar al depósito durante su ruta. Sin embargo, en el PVRP el horizonte temporal es de varios días, de manera que los vehículos realizan

sus trayectos diariamente y tienen la posibilidad de visitar a los clientes durante cualquiera de los días del horizonte temporal, uno o varias días, según su demanda. Los primeros en tratar el PVRP son Beltrami y Bodin (1974), que estudian una aplicación al problema de recogida de basuras con un período de planificación de 6 días y desarrollan tres heurísticas sencillas para su resolución. En Alegre (2002) se amplían considerablemente los horizontes de planificación considerados y se utilizan técnicas metaheurísticas como algoritmos meméticos, búsqueda en entorno variable, encadenamiento de trayectorias y búsqueda dispersa que ofrecen buenos resultados.

- SDVRP: Problema de Rutas de Vehículos con Carga Divisible (en inglés *Split Delivery VRP*). En el VRP clásico la carga de cada cliente es indivisible, exigiéndose que cada uno sea atendido de una vez y por un único vehículo. En el SDVRP se relaja esta condición y se permite que la carga de algunos clientes se *divida* entre varios vehículos si así se consigue una reducción en los costes totales. El problema fue tratado por primera vez por Dror y Trudeau (1989, 1990), que lo definieron de forma precisa y estudiaron su estructura, además de proponer un algoritmo heurístico de búsqueda local. Posteriormente, Dror et al. (1994) propusieron una modelización como problema de programación lineal y una serie de desigualdades válidas para mejorar dicha modelización. Mullaseril et al. (1997) y Sierksma y Tijssen (1998) presentaron algunas aplicaciones prácticas del problema a la vida real que relanzaron el interés por el SDVRP. Algunos métodos exactos fueron estudiados por Gueguen (1999) y Gendreau et al. (2003), este último incluyendo ventanas de tiempo, y Archetti et al. (2005) propusieron un algoritmo de búsqueda tabú.

- SVRP: Problema de Rutas de Vehículos Estocástico (en inglés *Stochastic VRP*). Es un VRP en el cual algunos de los datos del problema son aleatorios, es decir, siguen una determinada distribución de probabilidad pero a priori se desconocen. Puede ocurrir, por ejemplo, que no se sepa con certeza a qué clientes hay que servir, pudiendo aparecer cada uno de ellos con una cierta probabilidad. También las demandas de los clientes pueden ser aleatorias en vez de constantes fijas, e incluso el tiempo que un vehículo tarda en desplazarse de un cliente i a otro j (o análogamente el coste de viaje) puede ser una variable aleatoria. La primera vez que se consideraron datos estocásticos relacionados con un VRP fue en un trabajo de Tillman (1969), en el que se presenta una modificación del algoritmo de los ahorros de Clarke y Wright (1964) adaptado para considerar demandas distribuidas según una Poisson. En Stewart y Golden (1983) se realiza un estudio más profundo del SVRP con demandas estocásticas y se formula el problema introduciendo una función de penalización y utilizando restricciones que dependen de la realización de las variables aleatorias presentes en el problema. En Dror et al. (1989) también se pueden encontrar varios modelos alternativos que se basan en ideas similares. Otras referencias adicionales son Laporte et al. (1989), Carraway et al. (1990) y Lambert et al. (1993), donde se estudia el VRP con demandas estocásticas, y Gendreau et al. (1995, 1996), Roberts (1998) y

Roberts y Hadjiconstantinou (1998), donde se aborda el VRP con tiempos de viaje estocásticos.

- VRPPD: Problema de Rutas de Vehículos con Entrega y Recogida de mercancías (en inglés *VRP with Pickup and Delivery*). Es un VRP en el que, además de los clientes habituales que requieren la entrega de mercancías, existen algunos clientes que desean devolver cierta cantidad de mercancía. Para verificar que esta recogida se puede realizar hay que tener en cuenta que el vehículo que la realice debe tener suficiente espacio libre para acomodar la mercancía recogida, lo cual provoca que la planificación de los vehículos y la construcción de las rutas sea todavía más compleja, se precisen más vehículos y las distancias recorridas sean generalmente mayores. Los primeros trabajos sobre el VRPPD trataban de resolver problemas prácticos relacionados con el transporte de personas, y fueron llevados a cabo por Wilson et al. (1971), Wilson y Weissberg (1976) y Wilson y Covin (1977) con el objetivo de optimizar los sistemas de transporte de las ciudades de Haddonfield (Nueva Jersey) y Rochester (Nueva York). Durante los últimos años el desarrollo de la investigación sobre el VRPPD ha sido bastante paralelo al de otras variantes del VRP, concentrándose en análisis de algoritmos heurísticos debido a que la mayoría de los problemas reales relacionados son de gran tamaño. Sin embargo, las metaheurísticas diseñadas para esta clase de problemas no han conseguido producir resultados tan buenos como los obtenidos para otras variantes del VRP. Es muy importante la idea de intentar aprovechar el proceso en paralelo para acelerar la ejecución de los algoritmos y diseñar heurísticas híbridas que combinan diferentes técnicas para obtener mejores soluciones, como por ejemplo el trabajo de Toth y Vigo (1997), en el que se implementa una búsqueda tabú conjuntamente con un procedimiento de inserción de vértices en la solución y otro de mejora.

- VRPTW: Problema de Rutas de Vehículos con Ventanas de Tiempo (en inglés *VRP with Time Windows*). Es un VRP en el que las demandas de los clientes deben ser satisfechas dentro de determinados intervalos de tiempo, denominados *ventanas de tiempo*. La versión del problema más estudiada en la literatura es la denominada *versión fuerte*, en la que los vehículos no pueden llegar al destino más tarde de la hora fijada como tope bajo ningún concepto. En la *versión débil*, menos estudiada en la literatura pero más próxima a la realidad en algunos casos, las ventanas de tiempo se pueden violar pagando un determinado coste adicional. Cabe destacar que el problema de encontrar una solución factible para el VRPTW con un tamaño de flota fijo es por sí mismo NP-completo (este resultado es un corolario de un teorema demostrado por Savelsbergh (1985) para el TSP con Ventanas de Tiempo), lo cual refleja su gran complejidad. La gran dificultad inherente al VRPTW provocó que los primeros trabajos relacionados con él se limitasen a estudios de casos (Pullen y Webb, 1967, Knight y Hofer, 1968, Madsen, 1976). Sin embargo, se han desarrollado algoritmos exactos (Cook y Rich, 1999, Madsen et al., 1999) y también metaheu-

rísticas (Rochat y Taillard, 1995, Homberger y Gehring, 1999, Cordeau, Laporte y Mercier, 2000) que han ofrecido resultados prometedores.

El libro de Toth y Vigo (2002) es una recopilación reciente sobre los problemas de Rutas de Vehículos donde se puede encontrar la información básica de la mayoría de las variantes del VRP consideradas en la literatura y multitud de referencias. Por otro lado, en la página web http://neo.lcc.uma.es/radi-aeb/WebVRP/, dedicada por completo al VRP, se puede encontrar información adicional relativa a formulaciones, aplicaciones reales, instancias, mejores soluciones conocidas, etc., además de enlaces interesantes a otras páginas relacionadas.

1.1.3. Problemas de Rutas por Arcos (ARP)

Todos los problemas de rutas comentados anteriormente pertenecen a lo que se denominan problemas de rutas *por vértices*, ya que el objetivo, de una manera u otra, consiste en visitar una serie de localizaciones que pueden ser representadas por vértices en un grafo. Sin embargo, existen problemas de rutas que se sitúan en un ámbito muy distinto, en los cuales el objetivo no es visitar una serie de vértices sino recorrer una serie de arcos en un grafo. Estos problemas se denominan problemas de rutas *por arcos* y se conocen por sus siglas en inglés ARP: *Arc Routing Problems*.

La versión básica de este grupo de problemas, que se denomina Problema de Rutas por Arcos con Capacidades (y es conocida por sus siglas en inglés CARP: *Capacitated Arc Routing Problem*), consiste en encontrar un conjunto de rutas de longitud total mínima para una flota de vehículos con capacidades de manera que se recorran una serie de arcos de un grafo no dirigido. Este problema, que fue introducido por Golden y Wond en 1981, surge de forma natural en situaciones de la vida real en las que las calles de las ciudades (representadas por arcos) son las que realmente requieren un determinado servicio, y no lugares aislados, almacenes o fábricas (representados por vértices). Algunos de estos servicios pueden ser la recogida de basuras, el reparto diario de correo o la retirada de nieve, que por su propia naturaleza requieren recorrer determinados arcos de la red y no vértices concretos. Otras referencias adicionales en las que se tratan el CARP y otros problemas relacionados son Assad y Golden (1995), Belenguer y Benavent (2003), Benavent et al. (1992) y Lacomme et al. (2001).

1.2. Técnicas de relajación

Dados un conjunto de soluciones factibles X y una función f definida sobre dicho conjunto, un *problema de optimización* consiste en encontrar una solución factible del

conjunto X que optimice la función f dada. Para un problema de minimización, una solución $\bar{x} \in X$ se llama *solución óptima* o *mínimo global* si $f(x) \geq f(\bar{x})$, $\forall x \in X$. Un problema de maximización se puede transformar de forma obvia en un problema de minimización y viceversa. Si el problema considerado es difícil de resolver, para la obtención de cotas inferiores es habitual la consideración de relajaciones del problema más sencillas.

Definición 1. Sea (P): $\min\{f(x) \mid x \in X\}$. El problema (Q): $\min\{g(x) \mid x \in Y\}$ es una *relajación* de (P) si y sólo si $X \subset Y$ y $g(x) \leq f(x)$ $\forall x \in X$.

Por ejemplo, si en un problema de programación lineal entera se relajan las restricciones de integralidad se obtiene una relajación, denominada *Relajación lineal*, que es un problema de programación lineal continua y por tanto más sencillo de resolver que el problema inicial. En las siguientes secciones se introducirán brevemente otras técnicas de relajación utilizadas en este trabajo para la obtención de cotas inferiores y la mejora de las mismas, como son la Relajación Lagrangiana, la Descomposición de Dantzig-Wolfe y la generación de Planos de Corte.

1.2.1. Relajación Lagrangiana

Muchos problemas de optimización complicados son realmente problemas sencillos con un conjunto relativamente pequeño de restricciones complicadas que alteran su estructura y son las responsables de su gran complejidad. La Relajación Lagrangiana consiste en relajar dichas restricciones complicadas y penalizar su violación en la función objetivo a través de unos pesos determinados, obteniendo una relajación del problema inicial más sencilla de resolver. Fue introducida por Held y Karp (1970, 1971) en la implementación de un algoritmo exacto para el TSP con el que se obtuvieron resultados prometedores, y posteriormente ha sido utilizada con éxito en la resolución de otros problemas de optimización combinatoria. En Shapiro (1979a y 1979b) y más recientemente en Fisher (2004a, 2004b) se puede encontrar información detallada acerca de la Relajación Lagrangiana y multitud de referencias adicionales.

Definición 2. Sea (P) el siguiente problema de programación matemática:

$$\begin{aligned} \text{(P)}\ \min\ & z = cx \\ \text{s.a.}\ & Ax = b && (1.1)\\ & Dx \leq e && (1.2)\\ & x \in X && (1.3) \end{aligned}$$

donde x es $n \times 1$, c es $1 \times n$, A es $m \times n$, b es $m \times 1$, D es $k \times n$ y e es $k \times 1$. El problema $(\text{LR}_{u,v})$ $\min\{z_{u,v} = cx + u(Ax-b) + v(Dx-e) \,|\, x \in X\}$, con $u = (u_1, \cdots, u_m)$

y $v = (v_1, \cdots, v_k) \geq 0$ reales, es la Relajación Lagrangiana de (P), con multiplicadores u, v, con respecto a las restricciones (1.1) y (1.2).

El problema ($\mathrm{LR}_{u,v}$) es, en efecto, una relajación de (P), ya que la región factible de ($\mathrm{LR}_{u,v}$) es la misma que la de (P) con menos restricciones y si $x \in X$, $Ax = b$, $Dx \leq e$ entonces $Ax - b = 0$ y $Dx - e \leq 0$, y así $cx + u(Ax - b) + v(Dx - e) = cx + v(Dx - e) \leq cx$, ya que $v \geq 0$ y por tanto $v(Dx - e) \leq 0$.

De esta manera, para cualquier conjunto de multiplicadores u, $v \geq 0$, el valor óptimo de ($\mathrm{LR}_{u,v}$) es una cota inferior de (P), por lo que se tiene una cota inferior para cada conjunto de multiplicadores.

Puesto que el objetivo es obtener una cota inferior lo más alta posible, el mejor conjunto de multiplicadores para la Relajación Lagrangiana se obtiene resolviendo el denominado problema Dual Lagrangiano, (LD) $\max\{z_{u,v} | v \geq 0\}$, que posee ciertas propiedades estructurales que lo hacen abordable. En Held et al. (1974) y Fisher et al. (1975) se estudian estas propiedades y se proponen distintas estrategias para su resolución.

La elección de los multiplicadores y el problema Dual Lagrangiano guardan una estrecha relación con la descomposición de Dantzig-Wolfe, como se verá en la siguiente sección.

1.2.2. Descomposición de Dantzig-Wolfe

La descomposición de Dantzig-Wolfe, introducida en Dantzig y Wolfe (1960, 1961), se aplica a problemas de programación lineal en las que existe un pequeño grupo de restricciones generales en las que aparecen todas las variables del problema y el resto de restricciones se pueden dividir en grupos que involucran subconjuntos disjuntos de variables. Este tipo de problemas, que se dice que tienen una *estructura diagonal en bloque*, son de la forma siguiente:

$$
\begin{array}{lllllllllll}
(P) & \min & c^1x^1 + c^2x^2 + \cdots + c^rx^r & & & & & & & & \\
\\
\text{s.a.} & A^1x^1 & + & A^2x^2 & + & \cdots & + & A^rx^r & = & b^0 \\
 & B^1x^1 & & & & & & & = & b^1 \\
 & & & B^2x^2 & & & & & = & b^2 \\
 & & & & & \ddots & & & & \\
 & & & & & & & B^rx^r & = & b^r \\
 & & & & & & & x^1, \cdots, x^r & \geq & 0
\end{array}
$$

donde hay m_0 restricciones generales que afectan a todas las variables y para cada $j = 1, \cdots, r$ hay m_j restricciones que afectan a la variable (vectorial) x^j, que tiene n_j compo-

nentes. Así, b^0 es $m_0 \times 1$ y para cada $j = 1, \cdots, r$ se tiene que c^j es $1 \times n_j$, x^j es $n_j \times 1$, A^j es $m_0 \times n_j$, b^j es $m_j \times 1$ y B^j es $m_j \times n_j$.

La descomposición de Dantzig-Wolfe se basa en descomponer el problema (P) con estructura diagonal en bloque en r subproblemas más sencillos. Para ello supongamos que el poliedro $P^j = \{x^j \in \mathbb{R}^{n_j} | B^j x^j = b^j,\ x^j \geq 0\}$ asociado al conjunto de restricciones de la variable x^j es acotado para cada $j = 1, \cdots, r$ (el caso no acotado es análogo introduciendo direcciones extremas) y que el conjunto de vértices de P^j es $\{\bar{x}_1^j, \cdots, \bar{x}_{N_j}^j\}$. Entonces cada $x^j \in P^j$ se puede poner como combinación lineal convexa de los vértices de P^j de la siguiente forma:

$$x^j = \sum_{k=1}^{N_j} t_k^j \bar{x}_k^j, \quad \sum_{k=1}^{N_j} t_k^j = 1, \quad t_k^j \geq 0\ \forall k = 1, \cdots, N_j$$

Y así, el problema (P) es equivalente al Problema Maestro (MP) siguiente:

$$(MP) \quad \min \quad \sum_{k=1}^{N_1} (c^1 \bar{x}_k^1) t_k^1 + \sum_{k=1}^{N_2} (c^2 \bar{x}_k^2) t_k^2 + \cdots + \sum_{k=1}^{N_r} (c^r \bar{x}_k^r) t_k^r$$

$$\text{s.a.} \quad \sum_{k=1}^{N_1} (A^1 \bar{x}_k^1) t_k^1 \quad + \quad \sum_{k=1}^{N_2} (A^2 \bar{x}_k^2) t_k^2 \quad + \quad \cdots \quad + \quad \sum_{k=1}^{N_r} (A^r \bar{x}_k^r) t_k^r \quad = \quad b^0$$

$$\sum_{k=1}^{N_1} t_k^1 = 1, \quad \cdots \quad , \quad \sum_{k=1}^{N_r} t_k^r = 1, \quad t_k^j \geq 0\ \forall k, j$$

El problema (MP) se puede resolver de forma eficiente con un algoritmo de generación de columnas, donde las columnas a añadir en cada iteración son vértices de los poliedros $P^1, \cdots, P^r$. El algoritmo comienza con un subconjunto pequeño de vértices de cada poliedro y en cada iteración se generan nuevos vértices con coste reducido negativo para añadirlos al problema (MP). La generación de un vértice del poliedro P^j con coste reducido se reduce a la resolución de un problema de la forma $\min\{z_j | B^j x^j = b^j,\ x^j \geq 0\}$, donde la función objetivo z_j representa el coste reducido de x^j. En Barnhart et al. (1998) y Desaulniers et al. (2005) se puede encontrar información detallada acerca de los algoritmos de generación de columnas, además de multitud de referencias adicionales.

La descomposición de Dantzig-Wolfe y el problema Dual Lagrangiano guardan una estrecha relación: si las restricciones relajadas en la Relajación Lagrangiana son las restricciones de acople de la descomposición de Dantzig-Wolfe (las restricciones en las que aparecen todas las variables), los valores óptimos del problema Dual Lagrangiano y de la relajación lineal de la Reformulación de Dantzig-Wolfe coinciden, ya que en realidad

ambos problemas son duales (Geoffrion, 1974, Fisher, 1981). Además, los valores óptimos de las variables duales asociadas a las restricciones de acople del problema maestro de la Reformulación de Dantzig-Wolfe forman un conjunto óptimo de multiplicadores para las restricciones relajadas en la Relajación Lagrangiana (Magnanti et al., 1976). Estas propiedades permiten relacionar las cotas obtenidas y combinar ambos métodos en la resolución de problemas de optimización duros.

1.2.3. Planos de Corte

Un plano de corte es una restricción válida para un problema de optimización que se añade a la formulación de una relajación suya eliminando la solución óptima de dicha relajación y sin eliminar ninguna solución factible del problema original. El objetivo de la introducción de planos de corte es conseguir una sucesión de relajaciones cada vez mejores con las que obtener cotas inferiores próximas a la solución óptima del problema original.

Definición 3. Sean (P) $\min\{f(x_1, \cdots, x_n) \mid (x_1, \cdots, x_n) \in X\}$ y (Q) una relajación de P. Sea $x^* = (x_1^*, \cdots, x_n^*) \notin X$ una solución óptima de Q. Una desigualdad de la forma $a_1x_1 + \cdots + a_nx_n \leq b$ es un plano de corte si es válida para P, es decir, $a_1x_1 + \cdots + a_nx_n \leq b$ $\forall (x_1, \cdots, x_n) \in X$, y no es válida para x^*, es decir, $a_1x_1^* + \cdots + a_nx_n^* > b$.

Sean P y Q los problemas de la definición 3. El algoritmo estándar para la obtención de cotas inferiores de P con la introducción de planos de corte en la formulación de Q es el siguiente:

Algoritmo 1. Planos de Corte

1. Resolver el problema relajado Q, obteniendo una solución óptima x^*.
2. Si $x^* \in X$, FIN: x^* es solución óptima de P.
3. Encontrar uno o más planos de corte para x^* y Q, añadirlos a la formulación de Q y volver al paso 1. Si no se encuentra ningún plano de corte para x^* y Q, FIN: $f(x^*)$ es una cota inferior para el valor óptimo de P.

Además de la elección de la relajación inicial Q, la estrategia para generar los planos de corte del paso 3 es la que determina el funcionamiento del algoritmo de Planos de Corte (Algoritmo 1). Los algoritmos de generación de planos de corte se denominan habitualmente *algoritmos de separación*. Existen algoritmos de separación genéricos que permiten obtener planos de corte para cualquier problema de programación lineal entera, como por ejemplo los cortes de Gomory (Gomory, 1958), que fueron los primeros en aparecer en la literatura. Sin embargo, utilizando algoritmos de separación genéricos la convergencia del algoritmo

suele ser lenta, y la eficiencia del algoritmo de Planos de Corte depende del diseño de un algoritmo de separación, adaptado al problema a resolver, que sea capaz de generar planos de corte más fuertes, entendiendo que un plano de corte es más fuerte que otro si restringe más el espacio factible.

1.3. Heurísticas y Metaheurísticas

Los algoritmos heurísticos (o heurísticas) y algoritmos metaheurísticos (o metaheurísticas) son estrategias generales que sirven para resolver de forma aproximada, esto es no garantizan una solución óptima, problemas de optimización de tipología muy variada. Estos algoritmos se introducen debido a la dificultad de poder encontrar óptimos globales en tiempo razonable para numerosos problemas de optimización. Con estos procedimientos se obtienen, con un tiempo de cómputo reducido, soluciones factibles que están frecuentemente próximas a la optimalidad.

En Hansen y Mladenović (2000a, 2001a) se proponen varias propiedades deseables que debería tener toda metaheurística. Las más importantes son las siguientes:

- *Sencillez:* debe estar basada en principios simples y claros.
- *Precisión:* los pasos de la metaheurística deben formularse de forma precisa y clara.
- *Coherencia:* las operaciones realizadas en cada problema particular deben derivarse de forma natural de los principios de la metaheurística.
- *Eficacia:* los procedimientos asociados a la metaheurística deben proporcionar soluciones óptimas o cercanas a la óptima para la mayoría de casos particulares.
- *Eficiencia:* el tiempo de computación requerido para suministrar soluciones cercanas a la óptima debe ser moderado.
- *Robustez:* el rendimiento de la metaheurística debe ser consistente sobre una amplia variedad de casos.
- *Amigabilidad:* debe estar bien definida, expresarse claramente y ser fácil de entender y de usar.
- *Innovación:* las ideas aportadas por la metaheurística deben dar lugar a nuevas aplicaciones.

Las metaheurísticas se aplican a problemas de optimización para obtener soluciones factibles cercanas a la óptima. En las secciones siguientes se introducen las ideas básicas

de las heurísticas y metaheurísticas más importantes y más utilizadas en la resolución de problemas de optimización combinatoria, y en particular en problemas de rutas de vehículos.

1.3.1. Búsqueda Local. Estructuras de entornos

Una técnica muy utilizada para encontrar buenas soluciones en problemas de optimización complicados es la *búsqueda local*. En este procedimiento se parte de una solución inicial y en cada iteración se sustituye dicha solución por otra mejor hasta llegar a un mínimo local. Las soluciones candidatas a ser elegidas como nueva solución para la iteración siguiente (que se denominarán soluciones *vecinas*) se obtienen realizando ciertos movimientos o modificaciones sobre la solución actual, utilizándose de forma implícita o explícita una *estructura de entornos*:

Definición 4. Una *estructura de entornos* en un espacio de soluciones X es una aplicación

$$\begin{aligned} \Omega : X &\longrightarrow 2^X \\ x &\longmapsto \Omega(x) \equiv \text{entorno de } x \end{aligned}$$

que asigna a cada solución $x \in X$ un subconjunto $\Omega(x) \subset X$ del espacio de soluciones factibles denominado *entorno de* x. Las soluciones $x' \in \Omega(x)$ se llaman soluciones *vecinas* de x.

Cualquier métrica introducida en el espacio de soluciones X induce una estructura de entornos, pero existen estructuras de entornos no inducidas por ninguna métrica. Una estructura de entornos se puede definir de forma implícita determinando los movimientos que se pueden realizar sobre la solución de partida para obtener las soluciones vecinas, de manera que la estructura obtenida puede no venir inducida por ninguna métrica sobre el espacio de soluciones.

Definición 5. Sea Ω una estructura de entornos en un espacio de soluciones X. Una solución $\hat{x} \in X$ se dice que es un *mínimo local con respecto a la estructura de entornos* Ω si $f(x) \geq f(\hat{x}),\ \forall x \in \Omega(\hat{x})$.

Definición 6. Sea $\Delta = \{\Delta_k,\ k = 1, \cdots, n_e\}$ un conjunto finito de estructuras de entornos en el espacio de soluciones X. Una solución $\hat{x} \in X$ se dice que es un *mínimo local con respecto a las estructuras de entornos* Δ si $f(x) \geq f(\hat{x}),\ \forall x \in \Delta_1 \cup \cdots \Delta_{n_e}$.

A continuación se presentan algunas de las metaheurísticas más importantes utilizadas para resolver problemas de optimización duros, en las cuales, si no se especifica lo contrario, se considera que el problema a resolver es de la forma $\min\{f(x) \mid x \in X\}$.

1.3.2. Búsqueda en Entorno Variable

La Búsqueda en Entorno Variable (conocida por sus siglas en inglés VNS: *Variable Neighborhood Search*) es una metaheurística propuesta por Mladenovic (1995), Mladenovic y Hansen (1997) y Hansen y Mladenovic (1999, 2000a, 2001a). Utiliza varias estructuras de entornos distintas y cambia sistemáticamente entre ellas durante el proceso de búsqueda.

La idea de cambiar el entorno de búsqueda es simple y puede aplicarse a gran cantidad de problemas de optimización. Debido a esto y a sus buenos resultados su desarrollo ha sido muy rápido, apareciendo durante los últimos años multitud de variantes y extensiones aplicadas a diferentes problemas de la literatura. Dependiendo del tamaño y naturaleza del problema a resolver y de los recursos disponibles se han ido introduciendo nuevos elementos para adaptar la Búsqueda en Entorno Variable a las nuevas exigencias, intentando siempre mantener la simplicidad del esquema básico (Hansen y Mladenovic, 2001b, 2002, 2003).

Motivación de la Búsqueda en Entorno Variable

La mayor parte de las heurísticas basadas en búsqueda local utilizan una sola estructura de entornos. Esto provoca que, una vez alcanzado un mínimo local con respecto a la estructura de entornos utilizada, sea necesario modificar de alguna forma la solución actual para evitar el estancamiento en el mínimo local encontrado y poder así continuar la búsqueda del mínimo global. La Búsqueda en Entorno Variable, por contra, utiliza varias estructuras de entornos distintas, y se basa en tres hechos simples:

1. Un mínimo local con respecto a una estructura de entornos no es necesariamente mínimo local con respecto a otra estructura de entornos distinta.
2. Un mínimo global es también mínimo local con respecto a todas las estructuras de entornos.
3. En numerosos casos los mínimos locales están relativamente cerca, ya sea con respecto a la misma o a distintas estructuras de entornos.

El primero de estos hechos es claro, ya que los conjuntos de soluciones vecinas de una solución con respecto a estructuras de entornos distintas son diferentes en general, mientras que el segundo es consecuencia inmediata de la definición de mínimo global. El tercero es una importante conclusión empírica a la que se ha llegado después de observar la estructura de los espacios de soluciones de problemas muy variados, y que afirma que los óptimos locales están relacionados entre sí y pueden proporcionar información acerca del óptimo global. Generalmente no se conoce la relación que existe entre los óptimos del problema, locales o globales, ni las características que pudieran tener en común, por lo que

es conveniente analizar las proximidades de los óptimos locales encontrados para conseguir otros mejores.

La introducción de varias estructuras de entornos en la búsqueda local viene motivada por estos tres hechos relacionados entre sí, y es la idea central de los algoritmos de Búsqueda en Entorno Variable. Los distintos algoritmos propuestos del tipo VNS se diferencian en detalles tales como la técnica utilizada para generar soluciones iniciales o el criterio de parada, pero lo que principalmente distingue unos de otros, además de las estructuras de entornos utilizadas, son las reglas, determinísticas o estocásticas, que rigen los cambios entre dichas estructuras.

1.3.2.1. Búsqueda en Entorno Variable Descendente

La denominada Búsqueda en Entorno Variable Descendente (conocida por sus siglas en inglés VND: *Variable Neighborhood Descent*) es la Búsqueda en Entorno Variable más intuitiva. Se basa en la realización de un cambio en la estructura de entornos al llegar a un mínimo local, con la intención de escapar de él y obtener otro mínimo local mejor.

Fijada una estructura de entornos y una solución inicial, con una búsqueda local se encuentra la mejor solución perteneciente al entorno de la solución de partida. Un algoritmo de Búsqueda en Entorno Variable Descendente (VND) modifica la estructura de entornos *de forma determinística* cada vez que se alcanza un mínimo local, volviéndose a realizar una búsqueda local sobre la nueva estructura de entornos hasta la obtención de un nuevo mínimo local. Una de las estrategias de búsqueda local más sencillas y utilizadas, la búsqueda local *greedy*, reemplaza iterativamente la solución actual por la solución obtenida por la búsqueda local mientras ésta mejore el valor de la función objetivo, parando al llegar a un óptimo local.

El algoritmo termina cuando no existe ninguna solución que mejore la solución actual en ninguno de los entornos considerados. De esta manera, la solución obtenida con este procedimiento es mínimo local con respecto a todas las estructuras de entornos utilizadas, y aumenta la probabilidad de ser un mínimo global respecto a la que se tendría utilizando un único entorno.

El esquema general de un algoritmo VND se presenta a continuación.

Algoritmo 2. Búsqueda en Entorno Variable Descendente (VND)

1. *Inicialización:* Seleccionar una solución inicial S y un conjunto finito de estructuras de entornos $\{\Delta_k,\ k = 1, \cdots, n_e\}$ sobre el espacio de soluciones del problema. Poner $k = 1$.

2. *Búsqueda local en el entorno k-ésimo:* Encontrar la mejor solución $\hat{S} \in \Delta_k(S)$ del k-ésimo entorno de S.
3. *Cambio de entorno:* Si la solución $\hat{S}$ es mejor que S, poner $S = \hat{S}$ y $k = 1$; en otro caso, poner $k = k + 1$.
4. *Criterio de parada:* Si $k \leq n_e$ volver al paso 2. En otro caso, FIN: la mejor solución encontrada es $\hat{S}$.

En el Algoritmo 2 se reinicia la búsqueda desde la primera estructura de entornos cada vez que se mejora una solución. Sin embargo, este reinicio no es estrictamente necesario y la búsqueda se puede continuar utilizando la siguiente estructura de entornos, no volviendo a la primera hasta que no se hayan utilizado todas. El algoritmo pararía cuando se hubieran utilizado todas las estructuras de entornos, de la primera a la última, sin conseguir mejorar la solución. De esta manera se utilizan todas las estructuras el mismo número de veces, y la solución final sigue siendo mínimo local con respecto a todas las estructuras de entornos. El correspondiente pseudocódigo se presenta a continuación:

Algoritmo 3. VND sin Reinicio

1. *Inicialización:* Seleccionar una solución inicial S y un conjunto finito de estructuras de entornos $\{\Delta_k, k = 1, \cdots, n_e\}$ sobre el espacio de soluciones del problema. Poner $k = 1$ y $mej = \text{F}$.
2. *Búsqueda local en el entorno k-ésimo:* Encontrar la mejor solución $\hat{S} \in \Delta_k(S)$ del k-ésimo entorno de S.
3. *Actualización de la solución:* Si la solución $\hat{S}$ es mejor que S, poner $S = \hat{S}$ y $mej = \text{T}$.
4. *Cambio de entorno:*
   - Si $k < n_e$, hacer $k = k + 1$.
   - Si $k = n_e$ y $mej = \text{T}$, hacer $k = 1$, $mej = \text{F}$.
   - Si $k = n_e$ y $mej = \text{F}$, hacer $k = n_e + 1$.
5. *Criterio de parada:* Si $k \leq n_e$ volver al paso 2. En otro caso, FIN: la mejor solución encontrada es $\hat{S}$.

En la búsqueda local del paso 2 de los algoritmos 2 y 3 la solución $\hat{S}$ resultado de la búsqueda es la mejor solución del entorno actual, siendo ésta la solución que se envía al paso 3 para proceder a la actualización. Sin embargo, $\hat{S}$ no tiene porqué ser un mínimo local con respecto a la estructura de entornos que se esté utilizando, ya que $\Delta_k(\hat{S})$ no se explora y podría haber alguna solución en dicho entorno mejor que $\hat{S}$. Para conseguir que la solución $\hat{S}$ resultado de la búsqueda del paso 2 sea siempre mínimo local con respecto a la estructura de entornos actual, bastaría con realizar una búsqueda local greedy como la que se presenta a continuación en el paso 2'.

Paso 2'.

2.1. $\hat{S} = S$.

2.2. Encontrar la mejor solución $\bar{S} \in \Delta_k(\hat{S})$ del k-ésimo entorno de $\hat{S}$.

2.3. Si $\bar{S}$ es mejor que $\hat{S}$, poner $\hat{S} = \bar{S}$ y volver al paso 2.2.

1.3.2.2. Búsqueda en Entorno Variable Básica

En la Búsqueda en Entorno Variable Básica (conocida por sus siglas en inglés BVNS: *Basic Variable Neighborhood Search*) se introduce cierta aleatoriedad en la elección de soluciones, aunque los cambios de entorno se realizan de forma determinística. En cada iteración se elige al azar una solución en el entorno actual de búsqueda, la cual sirve como solución inicial para un proceso de búsqueda local.

El esquema general de un algoritmo BVNS se presenta a continuación.

Algoritmo 4. Búsqueda en Entorno Variable Básica (BVNS)

1. *Inicialización:* Seleccionar una solución inicial S y un conjunto finito de estructuras de entornos $\{\Delta_k,\ k = 1, \cdots, n_e\}$ sobre el espacio de soluciones del problema. Elegir un método de búsqueda local BL y una condición de parada CP.
2. *Reinicio:* Poner $k = 1$.
3. *Agitación:* Elegir de forma aleatoria una solución $\tilde{S} \in \Delta_k(S)$ del k-ésimo entorno de S.
4. *Búsqueda local:* Tomando $\tilde{S}$ como solución inicial, utilizar el método de búsqueda local BL para obtener el mínimo local $\hat{S}$.
5. *Cambio de entorno:* Si la solución $\hat{S}$ es mejor que S, hacer $S = \hat{S}$ y $k = 1$; en otro caso, poner $k = k + 1$.
6. Si $k \leq n_e$, volver al paso 3.
7. *Criterio de parada:* Si no se cumple la condición de parada CP, volver al paso 2. En otro caso, FIN: la mejor solución encontrada es S.

En la inicialización del algoritmo hay que elegir la solución inicial, las estructuras de entornos que se van a utilizar, el método de búsqueda local que va a seguirse en cada entorno (por ejemplo una búsqueda greedy) y la condición de parada. Esta última puede ser el máximo número de iteraciones, ya sean totales o entre dos mejoras de la solución, el máximo tiempo de cómputo disponible, etc.

La parte más importante y novedosa de este algoritmo es la *agitación de la solución* en el paso 3. Antes de realizar la búsqueda local, se *agita* el entorno de la solución actual y se elige al azar una solución perteneciente a él. A partir de ella, ya sea mejor o peor que la anterior, se inicia el proceso de búsqueda local. El objetivo de esta elección aleatoria es doble: por un lado se pretende alejarse del mínimo local del que se parte, intentando ampliar la búsqueda para encontrar otro mejor, mientras que por otro se evita el ciclado, que se podría producir con criterios determinísticos.

La búsqueda local del paso 4 también es crucial para el comportamiento del algoritmo. Puede elegirse una simple búsqueda greedy o algo mucho más complejo, como puede ser, incluso, una Búsqueda en Entorno Variable Descendente (VND). Dada su importancia, este último caso se estudia posteriormente.

El criterio de parada CP sólo se comprueba cuando se han recorrido todos los entornos sin conseguir mejorar la solución. Sin embargo, esto no implica que la solución S^* obtenida por el algoritmo sea necesariamente un mínimo local con respecto a todas las estructuras de entornos consideradas, ya que en la fase de agitación se generan soluciones nuevas cuyos entornos para valores de k menores que el actual no se llegan a explorar, pudiendo existir alguna solución en estos entornos mejor que S^*. Así, en la VNS Básica nunca se podrá garantizar que la mejor solución encontrada sea mínimo local con respecto a todas las estructuras de entorno utilizadas, y por ello no es imprescindible esperar a que se hayan recorrido todas las estructuras de entornos sin encontrar mejora para comprobar la condición de parada CP.

Variantes de la BVNS

Hay múltiples posibilidades para extender la Búsqueda en Entorno Variable Básica dotándola de elementos adicionales que puedan mejorar su comportamiento y eficiencia. A continuación se presentan algunas de las extensiones más utilizadas y que han ofrecido mejores resultados:

- La BVNS es un algoritmo de búsqueda descendente de primera mejora, ya que si se consigue mejorar la solución actual en el primer entorno explorado ésta se actualiza sin explorar el resto de entornos en busca de otra solución que produzca una mejora mayor. Puede transformarse fácilmente en un método *ascendente-descendente*, permitiendo que la solución $\hat{S}$ resultado de la búsqueda local sea aceptada como nueva solución actual con una determinada probabilidad, aunque sea peor que la solución actual o que la mejor solución encontrada hasta el momento. Así se favorece el movimiento de la solución entre mínimos locales distintos, además de introducir una nueva fuente de aleatoriedad que previene el ciclado. La probabilidad de aceptar una solución peor puede ir variando durante la ejecución del algoritmo, dependiendo del

número de iteraciones consecutivas sin mejora, de la calidad de la solución propuesta, etc.

- Una BVNS de mayor mejora explora en cada iteración todos los n_e entornos de la solución actual S y escoge la mejor solución encontrada entre todos ellos como la nueva solución actual. Esta extensión supone un coste computacional mayor, ya que habría que considerar todos los entornos en cada iteración, y no garantiza mejores resultados que la BVNS estándar, ya que, en general, no se puede asegurar que la versión de mayor mejora de un algoritmo se comporte mejor que la de primera mejora.

- En la fase de agitación, la solución de partida para la búsqueda local se genera de forma totalmente aleatoria, lo cual es bueno para evitar el ciclado y para alejarse del actual mínimo local, pero puede empeorar en exceso el valor de la función objetivo. Para evitar esto último dicha solución podría elegirse como la mejor de entre unas cuantas soluciones aleatorias en el entorno actual, cuyo número sería controlado por un parámetro que podría ser constante o variar durante la ejecución del algoritmo en función del número de iteraciones consecutivas sin mejorar la solución, del porcentaje de empeoramiento de la solución en las últimas iteraciones, etc.

- Como suele ser habitual, cuando se consigue mejorar la solución (paso 6) se reinicia el contador k de entornos a 1. Sin embargo, al igual que ocurría en la VND, este reinicio no es indispensable, y el algoritmo podría pasar a la siguiente estructura de entornos aunque se encuentre una solución mejor, volviendo a la primera solamente al finalizar un ciclo. Para ello basta eliminar el reinicio $k = 1$ del paso 6 y actualizar la estructura de entornos haciendo $k = k + 1$ independientemente de si $\hat{S}$ es mejor que S o no. Además, así se asegura que todas las estructuras de entornos se utilicen el mismo número de veces durante la búsqueda.

1.3.2.3. Búsqueda en Entorno Variable General

La Búsqueda en Entorno Variable General (conocida por sus siglas en inglés GVNS: *General Variable Neighborhood Search*) es un caso particular de la BVNS, en el que la búsqueda local del paso 4 es una VND. Recibe una atención especial por haber proporcionado los mejores resultados en múltiples situaciones prácticas. Algunas de estas aplicaciones se pueden encontrar en Andreatta y Ribeiro (2002), Brimberg et al. (2000), Canuto et al. (2001), Caporossi et al. (1999), Caporossi, Gutman y Hansen (1999), Caporossi y Hansen (2000), Hansen y Mlacenović (2001), Ribeiro y Souza (2002) y Ribeiro et al. (2002).

Utiliza dos conjuntos de estructuras de entornos, uno para la agitación de la solución y otro para la búsqueda local descendente, los cuales pueden tener estructuras comunes, e incluso ser iguales o estar uno contenido en el otro. El esquema general de un algoritmo GVNS se presenta a continuación.

Algoritmo 5. Búsqueda en Entorno Variable General (GVNS)

1. *Inicialización:* Seleccionar una solución inicial S y dos conjuntos finitos de estructuras de entornos $\{\Delta_k,\ k = 1, \cdots, n_1\}$, $\{\Omega_k,\ k = 1, \cdots, n_2\}$ sobre el espacio de soluciones del problema. Elegir una condición de parada CP.
2. *Reinicio en Δ:* Poner $k_1 = 1$.
3. *Agitación:* Elegir aleatoriamente una solución $\tilde{S} \in \Delta_{k_1}(S)$ del k_1-ésimo entorno Δ_{k_1} de S.
4. *Reinicio en Ω:* $k_2 = 1$
5. *Búsqueda local:* Encontrar la mejor solución $\hat{S} \in \Omega_{k_2}(\tilde{S})$ del k_2-ésimo entorno Ω_{k_2} de $\tilde{S}$.
6. *Cambio de entorno Ω:* Si $\hat{S}$ es mejor que $\tilde{S}$, poner $\tilde{S} = \hat{S}$ y $k_2 = 1$; en otro caso, poner $k_2 = k_2 + 1$.
7. *Criterio de parada en VND:* Si $k_2 \leq n_2$ volver al paso 5.
8. *Actualización mejor solución:* Si $\hat{S}$ es mejor que S, poner $S = \hat{S}$ y $k_1 = 1$; en otro caso, poner $k_1 = k_1 + 1$.
9. Si $k_1 \leq n_1$, volver al paso 3.
10. *Criterio final de parada:* Si no se cumple la condición de parada CP, volver al paso 2. En otro caso, FIN: la mejor solución encontrada es S.

En el paso 1 se eligen los datos iniciales del algoritmo: estructuras de entornos a dos niveles (uno para la agitación de la solución, Δ, y otro para la búsqueda local, Ω), solución inicial y criterio de parada. Entre los pasos 4 y 7 se realiza una VND, controlada por el índice k_2. En S se almacena la mejor solución encontrada hasta el momento.

1.3.2.4. Búsqueda en Entorno Variable Sesgada

La Búsqueda en Entorno Variable Sesgada (conocida por sus siglas en inglés SVNS: *Skewed Variable Neighborhood Search*) se basa en la Búsqueda en Entorno Variable Básica, incorporando un mecanismo que permite explorar zonas del espacio de soluciones alejadas de la solución actual que no han sido consideradas hasta el momento. Esta idea se apoya en una observación empírica: una vez encontrada la mejor solución en una región grande del espacio de soluciones, suele ser necesario alejarse bastante de esa región para encontrar una solución mejor. Así, en una SVNS se permite aceptar una solución peor que la actual como siguiente solución en la búsqueda si es suficientemente distinta de la solución actual y por tanto ofrece acceso a otra región alejada de la actual.

Las soluciones de un entorno alejado de una solución S pueden ser notablemente diferentes de S. Debido a este hecho, la SVNS puede degenerar de alguna manera en una VNS multiarranque consistente en realizar varias ejecuciones independientes del algoritmo desde soluciones iniciales distintas. Para intentar evitar esta situación conviene incorporar alguna penalización sobre las soluciones que sean excesivamente distintas.

Así pues se tienen dos objetivos contrapuestos: por un lado, es beneficioso aceptar soluciones muy distintas de la actual para poder explorar regiones alejadas de ella, pero por otro no conviene aceptar soluciones demasiado diferentes para que el algoritmo no degenere en una heurística multiarranque. Si f es la función objetivo del problema, una solución $\hat{S}$ se aceptará como solución actual en lugar de S cuando $f(\hat{S})$ sea un poco mayor que $f(S)$, pero no demasiado, y las características de $\hat{S}$ sean suficientemente diferentes de las de S, con el fin de evitar oscilaciones constantes de S a $\hat{S}$.

Para definir de forma precisa el algoritmo se introduce una función que mide la diferencia entre dos soluciones del problema. Tal función es de la forma $d : X \times X \longrightarrow \mathbb{R}$, y debe verificar $d(T,S) = d(S,T) \geq 0 \ \forall S, T \in X$ y $d(S,T) = 0 \Leftrightarrow S = T$.

Además de esa función de diferencia d se introduce un parámetro μ, de tal manera que la solución $\hat{S}$ sustituiría a la solución actual S en la siguiente iteración sólo si $f(\hat{S}) - \mu d(S,\hat{S}) < f(S)$. El parámetro μ sirve para ajustar la influencia que tiene la diferencia entre las soluciones implicadas en la elección de la solución para la siguiente iteración. Un valor adecuado del parámetro se debe determinar de forma experimental en cada caso concreto.

El esquema general de un algoritmo SVNS se presenta a continuación.

Algoritmo 6. Búsqueda en Entorno Variable Sesgada (SVNS)

1. *Inicialización:* Seleccionar una solución inicial S, un conjunto finito de estructuras de entornos $\{\Delta_k,\ k = 1, \cdots, n_e\}$ sobre el espacio de soluciones del problema, una función de diferencia $d : X \times X \longrightarrow \mathbb{R}$ y un parámetro μ. Elegir un método de búsqueda local BL y una condición de parada CP. Hacer $S^* = S$.
2. *Reinicio:* Poner $k = 1$.
3. *Agitación:* Elegir aleatoriamente una solución $\tilde{S} \in \Delta_k(S)$ del k-ésimo entorno de S.
4. *Búsqueda local:* Tomando $\tilde{S}$ como solución inicial, utilizar el método de búsqueda local BL para obtener el mínimo local $\hat{S}$.
5. *Actualización mejor solución encontrada:* Si $\hat{S}$ es mejor que S^*, poner $S^* = \hat{S}$.
6. *Cambio de entorno:* Si $f(\hat{S}) - \mu d(S,\hat{S}) < f(S)$, poner $S = \hat{S}$ y $k = 1$; en otro caso, poner $k = k + 1$.
7. Si $k \leq n_e$, volver al paso 3.

8. *Criterio de parada:* Si no se cumple la condición de parada CP, volver al paso 2. En otro caso, FIN: la mejor solución encontrada es S^*.

1.3.2.5. Otras variantes de la Búsqueda en Entorno Variable

Las variantes de la VNS presentadas en los apartados anteriores son las más importantes y más comúnmente utilizadas en la práctica, aunque también existen otras que pueden ser útiles en ciertos casos particulares. A continuación se describen brevemente algunas de estas variantes, como son la VNS Reducida, Paralela y con Descomposición.

La Búsqueda en Entorno Variable Reducida (conocida por sus siglas en inglés RVNS: *Reduced Variable Neighborhood Search*) es una VND en la cual no se realiza ninguna búsqueda local para la elección de la siguiente solución del entorno k-ésimo a considerar, sino que dicha solución se elige aleatoriamente dentro del entorno correspondiente. La RVNS es el procedimiento más sencillo de Búsqueda en Entorno Variable, y es útil para instancias muy grandes de problemas de optimización complicados para las cuales la realización de búsquedas locales es demasiado costosa.

La Búsqueda en Entorno Variable Paralela (conocida por sus siglas en inglés PVNS: *Parallel Variable Neighborhood Search*) consiste en paralelizar una o varias fases de los algoritmos VNS presentados con anterioridad. Se puede paralelizar directamente el proceso de búsqueda local en cada entorno o considerar varias soluciones dentro del mismo entorno y realizar búsquedas locales en paralelo a partir de cada una de ellas, permitiendo o no la transmisión de información relativa a la mejor solución encontrada hasta el momento.

La Búsqueda en Entorno Variable con Descomposición (conocida por sus siglas en inglés VNDS: *Variable Neighborhood Decomposition Search*) se basa en la descomposición del problema a resolver en dos partes, de manera que la VNS se pueda aplicar a dos niveles. De esta manera se introduce un proceso de aproximación sucesiva y las búsquedas locales no se realizan en el espacio completo sino en un subespacio de soluciones parciales.

1.3.2.6. Aplicaciones de la Búsqueda en Entorno Variable

El desarrollo de la Búsqueda en Entorno Variable desde que fuera propuesta hace algo más de una década (Mladenovic, 1995) ha sido muy rápido y ha proporcionado buenos resultados en multitud de problemas de muy diverso tipo. A continuación se presenta una breve recopilación de las aplicaciones más importantes en los campos de teoría de grafos, optimización combinatoria y optimización continua (Hansen y Mladenović (2001b) y Hansen et al. (2003)).

Problemas en teoría de grafos

Sorprendentemente, la VNS ha tenido gran éxito en el campo de la teoría de grafos, a pesar de tratarse de un área con la cual, a priori, no guardaba mucha relación. Se ha utilizado para resolver problemas de optimización de invariantes dentro de una familia de grafos, sirviendo para refutar conjeturas y establecer otras nuevas, algunas de las cuales han podido ser probadas formalmente poco después (Caporossi y Hansen, 2000).

La VNS se ha aplicado a problemas de grafos en los que las soluciones son árboles. El *problema de Steiner* (encontrar el subárbol de menor longitud que conecta todos los vértices terminales de un grafo) ha sido abordado a través de la VND en Martins et al. (2000) y Ribeiro et al. (2002), al igual que otras variantes suyas donde los grados de los vértices del subgrafo solución están acotados (Ribeiro y Souza, 2002). Otros problemas similares, como el *problema del árbol filogenético* (Andreatta y Ribeiro, 2002), el de determinar el subárbol que minimiza la suma de las longitudes de las aristas no incluidas en el grafo y el peso de los vértices no conectados (Canuto et al., 2001), o el de encontrar un subárbol de longitud mínima con exactamente k aristas (Mladenović y Urošević, 2001), también han sido abordados con éxito a través de algoritmos del tipo VNS.

También se ha aplicado la VNS en problemas de grafos en los que las soluciones son conjuntos de aristas. Entre ellos se encuentran el *problema del máximo subgrafo completo*, también llamado *clique* (Hansen, Mladenović y Urošević, 2001), o el *problema de corte máximo* (Festa et al., 2001). También se han utilizado algoritmos VNS en varias aplicaciones prácticas, como el diseño de tendidos de cables (Costa et al., 2001) o el diseño de redes para la distribución de petróleo (Brimberg et al., 2003).

Problemas de optimización combinatoria

La Búsqueda en Entorno Variable se ha aplicado a problemas de empaquetado, de localización y de rutas, que son problemas de gran importancia en el campo de la planificación logística. A continuación se presentan algunas referencias que relacionan la VNS con cada uno de estos tipos de problemas.

- Problemas de empaquetado: La VNS básica se aplica a la resolución del problema de empaquetado unidimensional (bin-packing problem, BPP), consistente en empaquetar un conjunto de objetos de distinto tamaño en el menor número posible de cajas o compartimentos respetando su capacidad máxima (Fleszar y Hindi, 2003).

- Problemas de localización: El *problema de la p-mediana* consiste en determinar las p ubicaciones de centros de servicio que, de entre un conjunto de m localizaciones posibles, minimicen la suma de las distancias de n usuarios a su centro de servicio más cercano. Este problema ha sido resuelto exitosamente utilizando una VNS básica (Hansen y Mladenović,

1997) y una VNS paralela (Crainic et al., 2001 y García López et al., 2001). También se ha aplicado la Búsqueda en Entorno Variable a otros problemas similares, como el *problema del p-centro* (Mladenović et al., 2003), que consiste en minimizar el máximo de las distancias de los n usuarios a sus centros de servicio más cercanos en lugar de la suma, y el *problema simple de localización de plantas* (SPLP, *Simple Plant Location Problem*) en el que el objetivo es decidir el número p de localizaciones a utilizar y al que se aplica una VNS reducida y una VNS con descomposición (Hansen, Mladenović y Pérez Brito, 2001).

- Problemas de rutas: La VNS ha sido aplicada con éxito al problema del viajante (TSP, *Traveling Salesman Problem*), a los problemas de rutas de vehículos (VRP, *Vehicle Routing Problem*) y al problema de rutas por arcos (ARP, *Arc Routing Problem*), así como a algunas de sus extensiones.

  - TSP: En Burke et al. (2001), Hansen y Mladenovic (1999) y Mladenovic y Hansen (1997) se utiliza una VNS con movimientos de distintos tipos para la definición de los entornos para resolver el TSP. En Silva et al. (2000) y Ochi et al. (2001), la VNS también se aplica al *problema del comprador* (TPP, *Traveling Purchaser Problem*), que es una extensión del TSP en la que, dada una partición de las ciudades, hay que visitar al menos una de cada subconjunto de la partición.

  - VRP: El VRP con ventanas de tiempo (VRPTW, VRP *with Time Windows*) ha sido abordado con una VNS y una VND (Bräysy, 2003 y Cordone y Calvo, 2001). Al VRPB (VRP *with Backhauls*) se le ha aplicado una VND (Crispim y Brandao, 2001) y una VNS (Mladenović y Hansen, 1997), restringiéndose esta última al caso particular de un único vehículo (TSPB, TSP *with Backhauls*).

  - ARP y otros problemas de rutas: El *problema del ciclo mediano* (MCP, *Median Cycle Problem*) ha sido abordado con un algoritmo de VNS hibridizado con una Búsqueda Tabú (Rodríguez et al., 2003). También se han utilizado algoritmos del tipo VNS para resolver problemas de rutas por arcos (Ghiani et al., 2000, y Hansen y Mladenović, 2000b y 2001b), en los que las rutas deben recorrer un conjunto de arcos dado.

Problemas de optimización continua

El *problema de programación bilineal* aplicado al problema del refinado en la industria del petróleo ha sido abordado con éxito con la VNS (Audet et al., 2003). También se utilizaron algoritmos del tipo VNS para resolver el *problema de optimización minimax continuo* (Hansen y Mladenović, 2001c y Mladenović et al., 2004) y el *problema múltiple de Weber* (Brimberg et al., 2000 y Brimberg y Mladenović, 1996), que es la versión continua del problema de la p-mediana (las p ubicaciones solución pueden elegirse en todo el plano).

1.3.2.7. Propiedades de la Búsqueda en Entorno Variable

A continuación se analiza de qué forma la metaheurística de Búsqueda en Entorno Variable se ajusta a las propiedades deseables de las metaheurísticas introducidas al principio del capítulo, resaltando sus principales ventajas (Hansen et al., 2003).

La VNS se basa en el principio claro y simple de cambiar sistemáticamente la estructura de entornos durante la búsqueda. La sencillez de esta idea permite incorporarla fácilmente a otros procedimientos heurísticos y que se aplique con éxito a la resolución de problemas muy diversos.

La definición de las estructuras de entornos y de las reglas que rigen los cambios entre dichas estructuras son los detalles que determinan los pasos principales de cualquier algoritmo de VNS, y deben ser precisos y claros.

Los procedimientos derivados de los distintos esquemas básicos de la VNS que aparecen en la literatura se derivan directamente de las ideas básicas de la metaheurística, las cuales parten del principio básico de utilizar varias estructuras de entornos para guiar la búsqueda.

La VNS proporciona soluciones óptimas o próximas a la óptima en gran variedad de problemas de optimización (Hansen et al., 2003) en tiempo computacional razonable, lo que muestra su eficacia y eficiencia.

Los problemas a los que se ha aplicado la VNS son de muy diversa naturaleza (Hansen y Mladenović, 2001b), no siendo necesario, además, realizar un ajuste extremadamente fino de los parámetros (si los hay) para conseguir resultados de calidad. Estos dos hechos muestran que se trata de una metaheurística robusta, cuyo comportamiento no se ve afectado drásticamente por las condiciones particulares del problema.

Los procedimientos básicos que se derivan de la VNS son claros e intuitivos, resultando fáciles de implementar y entender, además de contar, por regla general, con pocos parámetros que calibrar. Todo ello favorece la amigabilidad de la metaheurística y su aplicabilidad en distintos campos.

La capacidad de innovación de la Búsqueda en Entorno Variable está avalada por varias nuevas investigaciones motivadas por las ideas de esta nueva metaheurística: análisis de movimientos y estructuras de entornos y selección de los mismos, el sistema Auto-GraphiX (AGX) de descubrimiento científico por ordenador (Caporossi y Hansen, 2000), aplicaciones a la generación de columnas enfocada y estabilizada, etc.

Queda puesto de manifiesto, por tanto, que la Búsqueda en Entorno Variable posee las propiedades deseables citadas, lo cual justifica su importancia.

1.3.3. Búsqueda en Entornos Grandes

La Búsqueda en Entornos Grandes (conocida por sus siglas en inglés LNS: *Large Neighborhood Search*) es una metaheurística de búsqueda local que fue introducida por Shaw (1997) para la resolución de problemas de rutas de vehículos. Se basa en la eliminación de determinadas partes o elementos de una solución, obteniendo una solución parcial que posteriormente es reconstruida añadiendo nuevos elementos y rediseñando las partes faltantes con el objetivo de obtener una nueva solución mejor que la inicial. Existen multitud de formas distintas de reconstruir las soluciones parciales obtenidas en la LNS, que frecuentemente difieren mucho de la solución inicial considerada, y cada una da lugar a una solución vecina distinta, lo cual pone de manifiesto que esta metaheurística utiliza implícitamente entornos de gran tamaño, y de ahí su nombre, aunque se manejen de forma distinta a como se hace en los algoritmos de Búsqueda en Entorno Variable.

Para diseñar un algoritmo LNS hay que determinar cómo se eligen los elementos que se eliminan de la solución inicial, qué parte se mantiene fija y qué estrategias se pueden utilizar para reconstruir las soluciones parciales. Los entornos que habitualmente se manejan tienen grandes dimensiones, por lo que un proceso enumerativo no es factible y es necesario diseñar estrategias de reconstrucción potentes y adaptadas al problema concreto a resolver. A continuación se presenta un esquema general de un algoritmo de Búsqueda en Entornos Grandes.

Algoritmo 7. Búsqueda en Entornos Grandes (LNS)

1. *Estrategias de búsqueda:* Definir los operadores $P = \{P_1, \cdots, P_k\}$ que se utilizarán para la obtención de soluciones parciales mediante la eliminación de determinadas partes de las soluciones iniciales y los operadores $R = \{R_1, \cdots, R_m\}$ que se utilizarán para la reconstrucción de las soluciones parciales.

2. *Inicialización:* Seleccionar una solución inicial $S \in X$ y un criterio de parada CP. Poner $S^* = S$.

3. *Destrucción:* Elegir $P_i \in P$ y aplicar dicho operador para formar la solución parcial $P_i(S)$. Poner $S = P_i(S)$.

4. *Reconstrucción:* Elegir un operador $R_j \in R$ que sirva para reconstruir las soluciones parciales creadas con P_i y formar la solución factible $R_j(S)$. Poner $S = R_j(S)$.

5. *Actualización mejor solución:* Si $f(S) < f(S^*)$ poner $S^* = S$.

6. *Criterio de parada:* Si no se cumple la condición de parada CP, volver al paso 3. En otro caso, FIN: la mejor solución encontrada es S^*.

La LNS se aplicó a la resolución del VRPTW en Shaw (1997,1998) y Bent y Van Hentenryck (2004), dando lugar a buenos resultados, y posteriormente también fue aplicada al PDPTW por Bent y Van Hentenryck (2003). En Schrimpf et al. (2000) se propone una heurística similar a la LNS que se denomina *ruin and crecreate*. El Doble TSP con Múltiples pilas (DTSPMS), objeto de estudio de esta monografía, también fue abordado a través de un algoritmo LNS en Petersen y Madsen (2009).

1.3.4. GRASP

El GRASP (del inglés *Greedy Randomized Adaptative Search Procedure*) es una metaheurística de búsqueda adaptativa aleatorizada descrita explícitamente por primera vez por Feo y Resende (1989), aunque el nombre por el que se la conoce actualmente (GRASP) no fue introducido hasta 6 años después (Feo y Resende, 1995). Actúa en dos fases, una constructiva en la que se genera una solución inicial y otra de mejora local en la que se busca un mínimo local.

En la fase constructiva las soluciones se construyen de forma iterativa, añadiendo un elemento nuevo en cada iteración siguiendo una estrategia greedy aleatorizada similar a la introducida por Hart y Shogan (1987) para heurísticas semi-greedy: en cada iteración, el siguiente elemento que va a pasar a formar parte de la solución se elige de entre una lista de candidatos ordenada según el beneficio estimado que produciría cada uno. La heurística es adaptativa porque la estimación del beneficio que produce cada elemento se actualiza en cada iteración según se va construyendo la solución, y es aleatorizada porque no se añade necesariamente el mejor de los candidatos en cada momento, sino que se elige de forma aleatoria entre los primeros elementos de la lista.

En la segunda fase se realiza una búsqueda local sobre las soluciones construidas en la primera fase para intentar alcanzar un buen mínimo local. Este procedimiento se repite un cierto número de veces hasta alcanzar el criterio de parada.

A continuación se presenta un esquema general del algoritmo GRASP.

Algoritmo 8. GRASP

1. *Inicialización:* Elegir una constante entera $k > 0$, un procedimiento de búsqueda local BL y un criterio de parada CP. Poner $S = \emptyset$ y $z^* = \infty$.
2. *Lista Ordenada:* Crear una lista ordenada L con los elementos que pueden añadirse a S.
3. *Sorteo:* Elegir aleatoriamente un elemento $a \in L$ entre los k primeros de la lista L.
4. *Actualizar solución:* Añadir a al conjunto S.

5. *Iterar:* Si S no es una solución factible del problema, volver al paso 2.

6. *Búsqueda local:* Tomando S como solución inicial, utilizar el método de búsqueda local BL para obtener un mínimo local $\hat{S}$.

7. *Actualización mejor solución encontrada:* Si $f(\hat{S}) < z^*$, poner $S^* = \hat{S}$ y $z^* = f(\hat{S})$.

8. *Criterio de parada:* Si no se cumple la condición de parada CP, poner $S = \emptyset$ y volver al paso 2. En otro caso, FIN: la mejor solución encontrada es S^*.

En Li et al. (1994), Feo y Resende (1995), González (1996), Silviera (2001) y Pitsoulis y Resende (2001) se puede encontrar información más detallada sobre el GRASP, y en Festa y Resende (2001) se analiza y comenta la bibliografía relacionada con la metaheurística existente hasta el año 2001.

1.3.5. Temple Simulado

El Temple Simulado (conocido por sus siglas en inglés SA: *Simulated Annealing*) es una metaheurística que simula el comportamiento de un sistema físico multipartícula. Los primeros en aplicar los mecanismos del temple físico de la materia condensada a problemas de optimización combinatoria, poniendo de manifiesto las analogías existentes entre estos problemas y los de la mecánica estadística, fueron Kirkpatrick et al. (1983) y Ĉerný (1985), que presentaron de forma independiente algoritmos para resolver el TSP y el VRP.

Los elementos básicos del Temple Simulado se pueden encontrar en numerosas referencias: Aarts y Lenstra (1997), Díaz et al. (1996), Dowsland (1995), Dowsland et al. (2001), Glover et al. (2003), Koulamas et al. (1994), Suman y Kumar (2006) y Van Laarhoven y Aarts (1992), entre otros. En Van Laarhoven (1988), Aarts y Korst (1989) y Johnson (1990) se recogen resultados computacionales del Temple Simulado aplicado al TSP, y en Gendreau et al. (2002) se presentan algunas aplicaciones recientes del SA a problemas de rutas.

La aportación principal del Temple Simulado consiste en la utilización de una variable de control T, denominada temperatura, que permite la aceptación de una solución que empeore la función objetivo, con una cierta probabilidad, para tratar de escapar de un mínimo local. Al principio de la ejecución del algoritmo la temperatura es alta, y ésta se va descendiendo lentamente a medida que avanza el algoritmo. Cuando la temperatura es alta, la probabilidad de aceptar una solución con peor valor que la actual en la función objetivo es relativamente grande, permitiendo escapar fácilmente de un mínimo local en las iteraciones iniciales; a medida que la temperatura va descendiendo la probabilidad de aceptar una solución peor va disminuyendo, haciendo que el algoritmo se vaya estabilizando en torno a un mínimo local que se espera que sea bueno.

La probabilidad de aceptar una solución con peor valor en la función objetivo que la solución actual a una determinada temperatura viene dada por la *función de aceptación*.

Definición 7. La *función de aceptación* de un algoritmo de Temple Simulado es una función

$$\begin{array}{rcl} p_a : X \times X \times (0,\infty) & \longrightarrow & [0,1] \\ (\hat{S}, S, T) & \longmapsto & p_a(\hat{S}, S, T) \end{array}$$

donde $p_a(\hat{S}, S, T)$ es la probabilidad de aceptar la solución $\hat{S}$ como solución actual en lugar de S a temperatura T. La función de aceptación debe ser decreciente en T y en $f(\hat{S}) - f(S)$. Habitualmente, si $f : X \longrightarrow \mathbb{R}$ es la función objetivo a minimizar, la función de aceptación se define como

$$p_a\left(\hat{S}, S, T\right) = \exp\left(\frac{f(S) - f(\hat{S})}{T}\right),\ \forall\, \hat{S}, S \in X,\ f(\hat{S}) > f(S),\ \forall T \in (0,\infty)$$

Se observa que la función p_a cumple las propiedades anteriormente indicadas y, además, en el caso límite $T \to \infty$ la probabilidad de aceptar cualquier solución peor que la actual es 1, por lo que siempre se aceptaría dicho empeoramiento; por otro lado, en el caso límite $T = 0$ la probabilidad de aceptar un incremento en la función objetivo es 0, y por tanto nunca se aceptaría una solución peor que la actual, eliminándose el efecto del Temple Simulado.

También es crucial para el comportamiento del algoritmo la forma en que se disminuye la temperatura en cada iteración, determinada por la *función de enfriamiento*.

Definición 8. La *función de enfriamiento* de un algoritmo de Temple Simulado es una función

$$\begin{array}{rcl} E : (0,\infty) & \longrightarrow & (0,\infty) \\ T & \longmapsto & E(T) \equiv \text{temperatura iteración siguiente} \end{array}$$

donde $E(T)$ es la nueva temperatura del sistema en la siguiente iteración, después de haber estado a temperatura T. La función de enfriamiento E debe ser decreciente en T y debe cumplir que $E(T) < T$ para cada $T \in (0,\infty)$. Normalmente, por simplicidad, la función E se define como $E(T) = \alpha T$, donde $\alpha \in (0,1)$ es el *factor de enfriamiento*. Se observa que cuanto mayor sea el factor de enfriamiento α, más lento será el proceso de enfriamiento. El factor de enfriamiento se puede variar a lo largo de la ejecución del algoritmo para acelerar o ralentizar el proceso de enfriamiento.

A continuación se presenta el esquema de un algoritmo de Temple Simulado.

Algoritmo 9. Temple Simulado (SA)

1. *Inicialización:* Seleccionar una solución inicial $S \in X$, una estructura de entornos Δ sobre X, la temperatura inicial T, la función de aceptación p_a, la función de enfriamiento E, el número máximo de iteraciones N_{max} para cada temperatura y la condición de parada CP. Poner $S^* = S$ y $n = 1$.

2. *Solución vecina:* Generar una solución $\hat{S} \in \Delta(S)$ del entorno de S.

3. *Actualización de la solución actual:*

   - Si $f(\hat{S}) < f(S)$ poner $S = \hat{S}$.
   - Si $f(\hat{S}) \geq f(S)$:
     a) Generar aleatoriamente un número $u \in (0, 1)$.
     b) Si $u < p_a(\hat{S}, S, T)$ poner $S = \hat{S}$.

4. *Actualización mejor solución:* Si $f(S) < f(S^*)$ poner $S^* = S$.

5. *Actualización contador iteraciones:* $n = n + 1$

6. Si $n \leq N_{max}$ volver al paso 2.

7. *Enfriamiento:* Poner $T = E(T)$.

8. *Criterio de parada:* Si no se cumple la condición de parada CP, poner $n = 1$ y volver al paso 2. En otro caso, FIN: la mejor solución encontrada es S^*.

En la fase de inicialización (paso 1) se eligen las funciones y parámetros básicos del algoritmo. La elección de los parámetros de un algoritmo de Temple Simulado aplicado a un caso concreto es crucial para que el algoritmo funcione adecuadamente. La condición de parada CP que se chequea se puede definir de múltiples formas: puede ser el número máximo de estados por los que se puede pasar, la temperatura mínima que se puede alcanzar, el número máximo de iteraciones sin conseguir mejorar la solución actual, etc.

En el paso 2, la generación de una solución vecina se puede hacer al azar o mediante un proceso de búsqueda local. En el paso 3, si $f(\hat{S}) \geq f(S)$ la probabilidad de aceptar la solución S como la siguiente solución actual del algoritmo es $p_a(\hat{S}, S, T)$; para asegurar que la aceptación de la solución S se realiza de acuerdo con dicha probabilidad se utiliza un número aleatorio entre 0 y 1 y sólo se acepta si $u < p_a(\hat{S}, S, T)$. En el paso 4 se actualiza la mejor solución conocida, en el paso 5 se cuentan las iteraciones que se han realizado en el estado actual, en el paso 6 se comprueba si se ha llegado al final del ciclo actual y en el paso 7 se disminuye la temperatura y se inicia un nuevo ciclo. Al cumplirse la condición de parada CP el algoritmo termina, siendo S^* la mejor solución encontrada.

1.3.6. Búsqueda Tabú

La Búsqueda Tabú (conocida por sus siglas en inglés TS: *Tabu Search*) fue introducida por Glover (1989, 1990) y es una metaheurística que ha mostrado buen comportamiento en numerosos problemas. Realiza una búsqueda ascendente-descendente basada en una sola estructura de entornos que va siendo guiada por varias estructuras de memoria a corto y largo plazo, las cuales almacenan datos sobre los movimientos realizados en el pasado para no volver a mínimos locales ya visitados y evitar el ciclado.

Werra y Hertz (1989), Hertz y de Werra, (1991), Glover y Laguna (1993), Glover et al. (1993) y Soriano y Gendreau (1997) presentan una introducción a los algoritmos de Búsqueda Tabú. Dos referencias clásicas son Glover, Laguna et al. (1993), un capítulo de *Annals of Operations Research* dedicado por completo a la Búsqueda Tabú, y el libro de Glover y Laguna (1997), el cual, además de los conceptos y elementos básicos, presenta múltiples aplicaciones y material avanzado, incluyendo ideas que aún están pendientes de ser explotadas en toda su extensión. Otras referencias importantes son Gendreau (2002), Glover y Laguna (2002), Laporte y Osman (1996), Ribeiro y Hansen (2002) y Voss et al. (1999).

Las primeras aplicaciones de la Búsqueda Tabú a problemas de rutas, Willard (1989) y Pureza y França (1991), no dieron buenos resultados. Sin embargo, algunas implementaciones posteriores cosecharon un mayor éxito: Barbarosoglu y Ozgur (1999), Gendreau et al. (1994), Osman (1993), Rego (1998), Rego y Roucairol (1996), Rochat y Taillard (1995), Taillard (1993), Toth y Vigo (2003), Xu y Kelly (1996).

La Búsqueda Tabú es, en esencia, una búsqueda local en la que los últimos movimientos realizados se marcan como movimientos *tabú* y no pueden volver a realizarse durante un número determinado de iteraciones. Para la búsqueda local se utiliza una única estructura de entornos Δ; sin embargo, aunque los conjuntos de soluciones vecinas suelen manejarse habitualmente a través de la estructura de entornos, en el caso de la Búsqueda Tabú es más adecuado hablar del *conjunto de movimientos* que permiten obtener las soluciones vecinas de una determinada solución inicial.

Definición 9. Sea $S \in X$ una solución del problema. Se define un *movimiento* $m \in M(S)$ sobre la solución S como el conjunto de modificaciones a realizar sobre S que permiten la construcción de una solución vecina suya. $M(S)$ representa el *conjunto de movimientos* posibles sobre la solución S y para cada movimiento $m \in M(S)$ la solución $m(S) \in \Delta(S)$ representa la solución vecina de S que se obtiene realizando el movimiento m sobre S.

Para cada $\hat{S} \in \Delta(S)$ existe un movimiento $m \in M(S)$ tal que $\hat{S} = m(S)$, es decir, cualquier solución vecina se puede obtener a partir de la solución inicial con algún movimiento.

Un movimiento puede reducirse a una simple trasposición si las soluciones vienen representadas por permutaciones de un conjunto de índices, consistir en un 2-intercambio en el caso problemas de rutas, o ser algo mucho más complejo en otro tipo de problemas.

Definición 10. Dada una solución S y un movimiento $m \in M(S)$, se define el *valor* del movimiento m sobre la solución S, y se denota por $v_S(m)$, a la mejora o empeoramiento que produce en la función objetivo la solución vecina $m(S)$ obtenida por dicho movimiento:

$$v_S(m) = f\left(m\left(S\right)\right) - f\left(S\right), \quad \forall S \in X,\ \forall m \in M(S).$$

En un problema de minimización los movimientos con valor negativo producirán mejoras en la solución. Al realizar una búsqueda local, en cada iteración se sustituye la solución actual por una solución de su entorno, de manera que el proceso de búsqueda queda constituido por una sucesión de movimientos sobre la solución de partida. No parece conveniente repetir un mismo movimiento varias veces en un intervalo pequeño de iteraciones, ya que esto podría dar lugar a pasos atrás y a repeticiones en el recorrido. Para registrar los movimientos realizados en las últimas iteraciones del algoritmo se utiliza una *memoria a corto plazo*, que se encarga de almacenar los últimos movimientos realizados para evitar que se repitan pocas iteraciones más adelante. De aquí surge el concepto de *movimiento tabú*, que da nombre a la metaheurística:

Definición 11. Un movimiento $m \in M(S)$ se dice que es *tabú*, y se denota por $m \in T(S)$, si dicho movimiento ha sido realizado recientemente y por tanto no se puede volver a utilizar en las próximas D iteraciones. El número D de iteraciones en las que no puede utilizarse un movimiento tabú se denomina *duración*.

Una vez realizado un movimiento, éste entra a formar parte de la lista tabú y no puede volverse a utilizar durante las próximas D iteraciones. Esta prohibición es demasiado restrictiva, ya que alguno de los movimientos tabú podría dar lugar a una solución mejor que la actual, o incluso mejor que la mejor obtenida hasta el momento, pero ésta no sería alcanzada por la prohibición de utilizarlo. Es conveniente, por tanto, relajar esta prohibición y permitir, en algunos casos, la utilización de movimientos tabú.

Definición 12. La *función de aspiración* de un algoritmo de Búsqueda Tabú es una función

$$\begin{aligned} A : X &\longrightarrow \mathbb{R} \\ S &\longmapsto A(S) \equiv \text{nivel de aspiración de } S \end{aligned}$$

que asigna a cada solución $S \in X$ el umbral por debajo del cual se puede aceptar un movimiento tabú. Si S es la solución actual y $m \in T(S)$ un movimiento tabú, se permite la realización de dicho movimiento, aun siendo tabú, si se verifica la condición

$$f\left(m\left(S\right)\right) < A\left(S\right) \tag{1.4}$$

es decir, si el valor en la función objetivo de la solución $m(S)$ obtenida con el movimiento tabú está por debajo del umbral asignado por la función de aspiración A a la solución actual. La condición (1.4) se denomina *condición de aspiración* y el valor $A(S)$ *nivel de aspiración*.

La permisividad en la utilización de movimientos tabú dependerá, por tanto, de la función de aspiración elegida. Algunas funciones de aspiración comúnmente utilizadas son:

- $A(S) = f(S)$. Se acepta un movimiento tabú si la solución que se obtiene realizándolo es mejor que la solución actual. En este caso se eliminaría el efecto de la memoria a corto plazo, obteniéndose una búsqueda local estándar.
- $A(S) = 0.95 f(S)$. Se acepta un movimiento tabú si mejora la solución actual en al menos un $5\,\%$.
- $A(S) = f(S^*)$. Sólo se acepta un movimiento tabú si consigue mejorar la mejor solución obtenida hasta el momento S^*.

La memoria a corto plazo y los movimientos tabú proporcionan información acerca de los movimientos realizados recientemente, pero también es interesante registrar cuántas veces ha sido realizado cada movimiento desde el inicio del algoritmo. Para almacenar esta información se introduce una *memoria a largo plazo*.

Definición 13. Se denomina *frecuencia* de un movimiento m, y se denota por $fr(m)$, al número total de veces que ha sido utilizado dicho movimiento desde el comienzo de la ejecución del algoritmo hasta la iteración actual.

La memoria a largo plazo almacena las frecuencias de los movimientos realizados hasta el momento y permite introducir en la búsqueda una estrategia de *diversificación* o de *intensificación*, según se pretenda penalizar aquellos movimientos que han sido utilizados más a menudo en las iteraciones anteriores o favorecerlos, respectivamente. Para ello se asigna una penalización $p(m)$ a cada movimiento m, dependiendo de su frecuencia $fr(m)$. Cuanto mayor sea la frecuencia del movimiento mayor será la penalización en valor absoluto, siendo positiva en caso de una estrategia de diversificación y negativa en caso de una estrategia de intensificación.

A continuación se presenta el esquema de un algoritmo de Búsqueda Tabú.

Algoritmo 10. Búsqueda Tabú (TS)

1. *Inicialización:* Seleccionar una solución inicial $S \in X$, una estructura de movimientos factibles M sobre X, el número D de iteraciones que se mantienen los movimientos tabú, la función de aspiración A y la condición de parada CP. Poner $S^* = S$.

2. *Movimientos válidos:* Poner $M^* = \{m \in M(S) \,|\, m \notin T(S) \;\; \text{ó} \;\; f(m(S)) < A(S)\}$
3. *Vecinos:* Poner $V = \left\{\tilde{S} \in X \,|\, \tilde{S} = m(S),\, m \in M^*\right\}$
4. *Mejor vecino:* Elegir la mejor solución $\hat{S} \in V$ vecina de S y poner $S = \hat{S}$.
5. *Mejor solución:* Si $f(S) < f(S^*)$ poner $S^* = S$.
6. *Actualización:* Actualizar T y A.
7. *Criterio de parada:* Si no se cumple la condición de parada CP, volver al paso 2. En otro caso, FIN: la mejor solución encontrada es S^*.

En el paso 1 se eligen los elementos que definen el algoritmo. En el paso 2 se forma el conjunto de movimientos válidos, que dan lugar al conjunto de vecinos aceptados de la solución actual (paso 3). En el paso 4 se elige la mejor solución vecina de la actual y se toma como solución de la siguiente iteración. En el paso 5 se actualiza la mejor solución encontrada y en el 6 los elementos de la búsqueda tabú (lista tabú, función de aspiración, memoria a largo plazo si se utiliza, valor de los movimientos, etc.).

1.3.7. Otras heurísticas

En esta sección se introducen brevemente algunas heurísticas que, aunque no se han aplicado al DTSPMS en este trabajo, han sido utilizadas con éxito en otros problemas de diversos ámbitos, y en concreto en problemas de rutas de vehículos.

Los Algoritmos Genéticos (conocidos por sus siglas en inglés GA: *Genetic Algorithms*) fueron introducidos en Holland (1975), donde se propone por primera vez la adaptación de procesos biológicos como la selección natural a la resolución de problemas de optimización. Las soluciones del problema de optimización dado se representan como cadenas de bits, denominadas *individuos*, y forman lo que se denomina una *población* de soluciones. Los individuos de la población se *reproducen*, dando lugar a nuevos individuos (nuevas soluciones), y se les aplica un proceso de *selección natural* y *supervivencia* del más fuerte que hace que las mejores soluciones vayan permaneciendo en la población y las peores vayan desapareciendo. Si el algoritmo se implementa correctamente la población final obtenida tras iterar el proceso anterior contiene un conjunto de soluciones de buena calidad. La eficacia del algoritmo depende de la correcta representación de las soluciones como cadenas de bits y de la implementación de los operadores de reproducción y selección. Información adicional detallada puede encontrarse en Goldberg (1989), que es una referencia clásica para algoritmos genéticos, y en Beasley et al. (1993), Whitley (1994) y Mitchell (1996). En Goldberg et al. (1997) se presenta una bibliografía muy extensa (con más de 4.000 entradas) sobre algoritmos genéticos, que también está disponible en la web ftp://ftp-illigal.ge.uiuc.edu/pub/papers/IlliGALs/97011.ps.Z.

Los Algoritmos de Colonias de Hormigas (conocidos por sus siglas en inglés ACO: *Ant Colony Optimization*) fueron introducidos en 1991 por M. Dorigo (Dorigo et al., 1991, Dorigo, 1992, Dorigo et al., 1996) como una nueva alternativa para la resolución de problemas de optimización duros. Se inspiran en el comportamiento de las colonias reales de hormigas durante la búsqueda de alimentos: al principio las hormigas exploran las zonas próximas al hormiguero de forma aleatoria, pero a medida que van encontrando comida depositan cierta cantidad de feromona a lo largo del camino de vuelta, dependiendo de la cantidad y calidad de la comida encontrada, para guiar a las demás hormigas y ayudarles a seguir los caminos más cortos desde el hormiguero hacia la fuente de comida. La aplicación más natural de esta estrategia son los problemas de rutas, y por ello fue aplicada en primer lugar al TSP (Dorigo et al., 1991 y 1996), produciendo resultados prometedores. Posteriormente también fue aplicada a otros problemas de optimización combinatoria como problemas de asignación (Costa y Hertz, 1997, Maniezzo y Colorni, 1999, Stützle y Hoos, 2000), de planificación (Gagné et al., 2002, Merkle et al., 2002, Blum y Sampels, 2004) y de rutas de vehículos (Gambardella et al., 1999, Reimann et al., 2004), entre otros. En Dorigo y Stützle (2004) se recopilan otras muchas aplicaciones de los algoritmos de colonias de hormigas y se puede encontrar información adicional sobre ellos.

La Búsqueda Dispersa (conocida por sus siglas en inglés SS: *Scatter Search*) fue introducida en Glover (1977) como una heurística para la resolución de problemas de programación entera. Parte de un conjunto pequeño de soluciones diversas, que se denominan puntos de referencia, y genera sistemáticamente combinaciones de dichos puntos de referencia para obtener soluciones nuevas de mayor calidad. Las combinaciones son formas generalizadas de combinaciones lineales, acompañadas de procesos adaptativos para garantizar condiciones de factibilidad. En Glover (1998), considerada la referencia básica de la Búsqueda Dispersa, se recogen y simplifican muchas de las ideas expuestas en trabajos anteriores. En Glover et al. (2000) se recopilan algunas de las implementaciones más importantes de la Búsqueda Dispersa a problemas de optimización combinatoria y se estudia su relación con la estrategia de Encadenamiento de Trayectorias (conocida por sus siglas en inglés PR: *Path Relinking*), en la que se interpreta el proceso de generar combinaciones lineales de un conjunto de referencia como el proceso de generar caminos entre estas soluciones. También puede encontrarse gran cantidad de información adicional sobre la Búsqueda Dispersa en el libro monográfico de Laguna y Martí (2003).

Capítulo 2

Planteamiento y descripción del DTSPMS

El *Doble Problema del Viajante con Múltiples Pilas* (al que se hace referencia con las siglas DTSPMS, del inglés *Double Traveling Salesman Problem with Multiple Stacks*) fue introducido en Petersen (2006) a raíz de un proyecto de colaboración de la Universidad Técnica de Dinamarca con una empresa de software informático. La compañía que planteó el problema por primera vez se dedica al diseño de software para la organización de flotas y rutas de transporte de diferentes empresas del sector logístico, por lo que se ve obligada a tratar problemas reales en constante evolución y con características muy distintas.

Su interés se centraba en encontrar la forma óptima de atender una serie de encargos consistentes en recoger cierta mercancía en determinadas localizaciones de una región y entregarla en las correspondientes localizaciones de otra región alejada geográficamente de la primera. La carga a transportar consiste normalmente en una cierta cantidad de euro-palés de tamaño estándar, que aunque pueden contener mercancías de diverso tipo, tienen unas dimensiones fijas que facilitan el uso de distintos medios de transporte. Los vehículos de la flota que posee la empresa logística son de carga trasera (rear-loaded) y disponen de un contenedor de 40 pies en el que se pueden almacenar 33 euro-palés si se colocan formando 3 filas de tamaño 11.

Por razones de seguridad, los conductores de los vehículos no están autorizados a manipular la carga transportada en ningún caso, por lo que no es posible ningún tipo de reorganización durante todo el proceso. Debido a esto y al hecho de que el vehículo es de carga trasera, durante la ruta de entrega los únicos palés accesibles en cada fila son aquellos que se han recogido en último lugar. Así, el container del vehículo queda dividido en 3 filas independientes con capacidad para 11 euro-palés, cada una de las cuales sigue un orden LIFO (Last-In-First-Out).

En esta situación, la empresa logística debe decidir qué ruta seguir para realizar la recogida de mercancía en la primera región, de qué manera almacenar dicha mercancía en el contenedor del vehículo y qué ruta seguir para entregar la mercancía en la segunda región, teniendo en cuenta las restricciones de espacio y maniobrabilidad del vehículo. El objetivo final es minimizar el coste total de la operación, que depende casi exclusivamente de la longitud de las rutas a cubrir.

El DTSPMS es un problema que incluye elementos novedosos con respecto a los problemas de rutas clásicos sobre los que tanto se ha investigado durante las últimas décadas, y por ello apenas ha sido aún tratado en la literatura especializada. En la Sección 2.1 se describe detalladamente el DTSPMS, se tratan algunos casos particulares y se presentan diversas peculiaridades y aplicaciones del problema. En la Sección 2.2 se propone una formulación como problema de programación matemática y en la Sección 2.3 se introducen otros problemas de rutas de vehículos presentes en la literatura y relacionados con el DTSPMS.

2.1. Planteamiento del problema

En el DTSPMS hay un único vehículo en el que la carga se puede colocar en varios compartimentos, en lo sucesivo llamados *pilas*, que tiene que recoger y repartir mercancías en dos regiones independientes, almacenando la mercancía recogida a los clientes de la primera región en las pilas disponibles y entregándola posteriormente a los clientes de la segunda región, sin posibilidad de recolocar los artículos almacenados una vez recogidos. La recogida y entrega de mercancía se realizan de forma independiente cada una en una región, pero el orden de recogida condiciona el posible orden de entrega, ya que cada pila obedece un principio LIFO de manera que si un artículo a es recogido antes que otro b y ambos se almacenan en la misma pila, el artículo b debe ser repartido antes que el a durante el proceso de entrega.

Los elementos del problema son *dos regiones* independientes, cada una modelizada por un grafo completo, y un conjunto de *encargos*, cada uno de los cuales consiste en recoger cierta mercancía en una determinada localización de la primera región y entregarla en otra localización de la segunda región. Cada región tiene además un depósito central del cual parte el vehículo y al cual regresa después de realizar su ruta.

En los grafos no dirigidos que modelizan las regiones del problema cada localización está representada por un vértice y cada conexión por una arista, de manera que cada encargo está asociado a un vértice en el grafo de recogida y a otro en el grafo de entrega. Así, el vehículo parte vacío del depósito central de la región de recogida, visita las distintas localizaciones de dicha región recogiendo la mercancía requerida y regresa al depósito central. Desde éste se transporta la carga recogida sin modificación alguna hasta el depósito

central de la región de entrega, desde donde parte otro vehículo cargado con esta mercancía para llevar cada artículo a su destino y regresar vacío al depósito central. En la Figura 2.1 se presentan las rutas de recogida y entrega de una solución de un DTSPMS con 6 encargos. El trayecto entre los depósitos de ambas regiones es ajeno al problema y por tanto no influye en su modelización ni en su resolución.

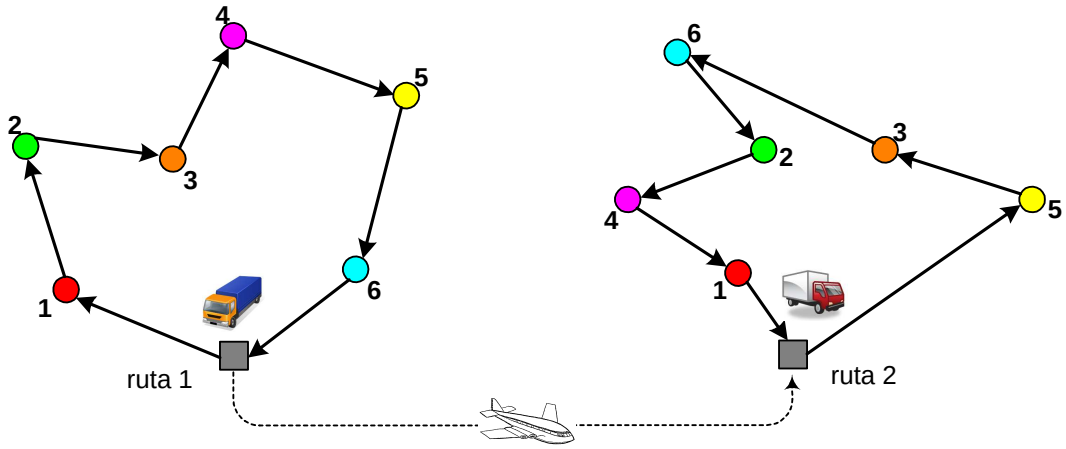


Figura 2.1: Una pareja de rutas factibles para un DTSPMS con 6 encargos

La zona de carga del vehículo consta de varias pilas, de manera que cuando se recoge un paquete se puede almacenar en la parte superior de cualquiera de las pilas con espacio libre, pero sin posibilidad alguna de reorganizar los paquetes recogidos anteriormente. Así, cuando el vehículo finaliza el recorrido de recogida la carga queda estructurada en varias pilas, apareciendo en sus cabeceras los últimos paquetes almacenados en cada una. Esta carga se transporta intacta hasta el depósito de la región de entrega, desde el cual comienza el reparto. Los únicos paquetes accesibles en cada momento son los situados en las cabeceras de las pilas, que siguen un principio LIFO; sin embargo, aunque cada pila de forma individual se ve sometida a esta restricción, globalmente la carga no sigue un principio LIFO, ya que las distintas pilas son independientes.

En el modelo básico se supondrá que todas las pilas tienen la misma capacidad, que todos los paquetes a recoger tienen las mismas dimensiones y que cada encargo consta de un único paquete. Tanto el número de pilas disponibles como su capacidad son datos del problema.

Las pilas representadas en todas las figuras de esta memoria se cargarán y vaciarán por la parte superior, de manera que los primeros encargos recogidos se van situando en las posiciones más bajas y los demás se van colocando encima; una vez llenas las pilas, el reparto de encargos comienza por arriba y en cada momento los únicos encargos disponibles para ser repartidos son aquellos situados en las cabeceras de las pilas. En la Figura 2.2 se presenta una asignación de pilas compatible con las rutas de la Figura 2.1. El número de

pilas disponibles es 2 y su capacidad máxima 3.

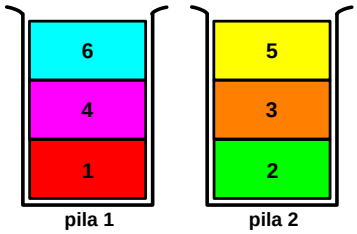


Figura 2.2: Una asignación de pilas factible para un DTSPMS con 6 encargos

El objetivo del problema consiste en encontrar dos ciclos hamiltonianos, uno en cada grafo, tales que la suma de sus longitudes sea mínima y cumplan las restricciones de capacidad (tamaño máximo de cada pila) y precedencia (regla LIFO en cada pila) establecidas anteriormente. Así, una solución factible del problema queda definida por una *ruta de entrega*, una *ruta de recogida* y una *asignación de pilas*. Las rutas indican el itinerario que sigue el vehículo en cada región y la asignación de pilas indica qué pila almacena cada paquete. Dicha asignación de pilas por sí sola no determina la posición de los paquetes en las pilas, sino que solamente indica la pila correspondiente a cada paquete; sin embargo, fijada una de las rutas, ya sea la de entrega o la de recogida, y una asignación de pilas, existe una única posición para cada paquete en la pila a la que está asignado que dé lugar a un *plan de carga* compatible con lo anterior.

2.1.1. Casos particulares

En los casos extremos del número de pilas la asignación de pilas resulta irrelevante. Si el número de pilas disponibles es igual al número de encargos en realidad no hay restricciones de precedencia, ya que cada paquete podría colocarse en una pila distinta, y el problema se reduce un TSP independiente en cada grafo. Si por el contrario el vehículo sólo dispone de una pila, la carga en su totalidad debe seguir una regla LIFO y por tanto las dos rutas de la solución tienen que ser exactamente inversas la una de la otra. En éste último caso, fijando una ruta se tiene la otra, reduciéndose el problema a un TSP cuya matriz de costes es la suma de las matrices de costes de los grafos de entrega y recogida del correspondiente DTSPMS.

Ambos casos extremos permiten acotar el coste de la solución óptima del problema original. El valor óptimo del problema con tantas pilas como encargos es una cota inferior para el DTSPMS estándar, mientras que cualquier solución del problema con una única pila es una solución factible del DTSPMS y proporciona una cota superior.

2.1.2. Campos de aplicabilidad del DTSPMS

Durante los últimos años el mundo de la logística y el transporte ha sufrido importantes y numerosos cambios que han conformado una situación actual muy diferente a la existente poco tiempo atrás. Constantemente surgen problemas nuevos, con aplicación directa en la vida real, que guardan cierta relación con otros problemas de optimización combinatoria ya estudiados con detalle en la literatura, pero la mayoría de ellos aportan elementos nuevos que los distinguen de los problemas clásicos y que los hacen interesantes en sí mismos. Además, algunos son extremadamente difíciles de resolver a pesar de ser conceptualmente simples, como es el caso del DTSPMS, lo cual aporta un atractivo añadido al problema.

El DTSPMS surge como aplicación directa de un problema logístico en una empresa de software informático que consistía en encontrar la forma óptima de atender una serie de recogidas y entregas de mercancías, normalmente euro-palés, entre clientes muy alejados geográficamente. Las dimensiones de los vehículos disponibles y la imposibilidad de manipular la carga impuesta por medidas de seguridad añaden una serie de restricciones al problema que le dan una estructura peculiar, aumentando su complejidad pero permitiendo al mismo tiempo, como se comenta a continuación, que se adapte fácilmente a otros problemas reales y pueda tener diversos campos de aplicabilidad.

Utilización de instrumentos robóticos

En ciertas situaciones reales del mundo de la industria la manipulación de determinadas piezas se realiza con instrumentos robóticos, en concreto brazos mecánicos que pueden recoger o colocar de forma automatizada una serie de piezas en distintos lugares. Las piezas recogidas de forma automatizada en una nave son almacenadas en un contenedor y trasladadas al lugar en que deben ser colocadas, también de forma automatizada. El contenedor suele estructurarse en varias pilas, y la utilización de brazos mecánicos automatizados hace que no sea posible la reorganización de la carga y que las piezas deban extraerse secuencialmente de cada pila. El diseño de secuencias eficientes de recogida y reparto de piezas para los brazos mecánicos es crucial para minimizar el tiempo total de la operación y el coste total de la misma, que habitualmente es directamente proporcional a la cantidad de desplazamientos que deben realizar los brazos mecánicos. Esta situación puede modelizarse como un DTSPMS en el que las rutas de recogida y entrega son las secuencias de recogida y reparto de los brazos mecánicos y la asignación de pilas indica cómo colocar las piezas en el contenedor.

Reorganización de la carga en problemas de rutas

Uno de los elementos importantes del DTSPMS es la imposibilidad de reorganizar la carga en los distintos compartimentos antes de ser repartida. Esta restricción puede venir originada por la prohibición, impuesta al conductor del vehículo, de manipular la carga, debido a la naturaleza de los materiales a transportar, o simplemente por razones

de responsabilidad, seguridad y garantía, ya que el cliente que entrega la mercancía la deposita en el contenedor del vehículo y sólo el cliente receptor está autorizado a extraerla para evitar posibles manipulaciones o deterioros del producto. Esta restricción sobre la carga también puede ser debida a una gran dificultad para mover los paquetes recogidos y por tanto recolocarlos (mercancía muy pesada o muy difícil de colocar convenientemente en el vehículo para ser transportada con seguridad), a la no disponibilidad de tiempo suficiente entre las visitas a los clientes para realizar adecuadamente la reorganización de la carga, o bien los costes ocasionados por la manipulación de la carga (mano de obra, maquinaria, tiempo...) podrían superar los costes extra que surgen al realizar las entregas en el orden impuesto por las pilas y no sería rentable la reorganización intermedia de la carga.

Agrupación de clientes en dos zonas

Una de las características más novedosas del problema es la separación de los lugares de recogida y entrega en dos regiones distintas muy alejadas geográficamente, de manera que haya que visitar primero todas las ciudades de la primera región para recoger los pedidos, transportar la carga hasta el depósito central de la segunda región y desde allí realizar todas las entregas. Cualquier empresa de mensajería con una cantidad suficientemente grande de clientes, que permita la agrupación de clientes por zonas, tiene que solventar situaciones similares cada día.

La separación de los clientes en dos regiones independientes permite utilizar en cada región el medio de transporte más adecuado. El vehículo elegido para visitar a los clientes de cada región debe ser capaz de transportar el contenedor con la carga, que será idéntico en los dos casos, pero en función de las distancias a recorrer, el tiempo meteorológico o las características de las carreteras en cada zona se puede elegir el vehículo cuyas prestaciones (tamaño, potencia, autonomía, etc.) se adapten mejor a las necesidades concretas. Por otro lado, el transporte del contenedor del depósito central de la región de recogida al depósito central de la región de entrega (*long-haul transport*) puede realizarse utilizando cualquier medio de transporte (camión, tren, avión, barco, etc.), dependiendo de la situación geográfica y la distancia que los separe.

Introducción de múltiples pilas

Otro elemento importante del problema es la existencia de varias filas o compartimentos en el vehículo que funcionan como pilas con un orden LIFO. En la práctica la utilización de varias pilas es posible si las dimensiones del contenedor del vehículo permiten el almacenamiento de los paquetes en varias filas independientes a lo ancho de su estructura, y con ello se consigue mayor flexibilidad a la hora de recoger los pedidos. En el caso de la aplicación real que motivó el problema, los elementos que debían ser transportados son los denominados *euro palés*, los cuales se usan comúnmente para el transporte de mercancías. Según las dimensiones estándar de los euro palés, estos se podían acomodar en contenedores de 40 pies de ancho formando 3 filas independientes de 11 elementos cada una. Por ello,

en las primeras instancias del DTSPMS que se abordaron había 33 encargos que podían almacenarse formando 3 pilas de tamaño 11. Posteriormente, también se consideraron instancias con 66 encargos, almacenados en contenedores de mayor tamaño o altura formando 3 pilas de tamaño 22.

Agrupación de encargos

Puede suceder que el número de localizaciones de recogida y entrega no coincidan, debido a la existencia de clientes que quieren enviar mercancías a distintos destinos o clientes que reciben encargos desde varias localizaciones diferentes. Es típico el caso en el que desde unos centros de producción se envían palés con distintos tipos de mercancías a varios destinos dentro de una misma zona; esta situación da lugar a un DTSPMS en el que el número de localizaciones de recogida (correspondientes a los centros de producción, que suelen ser pocos) es mucho menor que el número de localizaciones de entrega (correspondientes a los pequeños clientes que reciben mercancías desde los centros de producción). En Côté et al. (2009) se menciona una aplicación al reparto de muebles a domicilio de un problema similar al DTSPMS: primero se visitan los centros de producción y/o almacenes para recoger el mobiliario que se debe entregar, el cual se almacena en el contenedor del vehículo formando varias filas de productos, y posteriormente el vehículo se traslada a la zona urbana donde están los destinatarios finales para repartir los muebles encargados; los productos recogidos son habitualmente frágiles y pesados, por lo que su manipulación dentro del contenedor no está permitida y en cada momento sólo es accesible la mercancía almacenada en cada fila en último lugar.

También puede darse el caso opuesto, en el que gran número de clientes de una misma zona envían mercancías de varios tipos que deben ser tratadas en unos pocos centros de procesamiento. Desde el punto de vista de la resolución del problema ambas situaciones son equivalentes, ya que si se invierten los papeles de los clientes de recogida y entrega la estructura del problema se mantiene.

Una instancia del DTSPMS con distinto número de clientes de recogida y entrega se puede transformar fácilmente en una instancia equivalente del DTSPMS estándar con el mismo número de clientes en ambas zonas añadiendo tantos clientes ficticios como envíos o recogidas extra exija cada cliente. Las distancias entre los nuevos clientes ficticios y el original serán nulas, mientras que las distancias entre los nuevos clientes ficticios y los demás se mantendrán como en el problema original. De esta manera se obtiene una instancia del DTSPMS estándar cuyas soluciones pueden transformarse de forma sencilla en soluciones válidas para la instancia original. Cabe esperar, no obstante, que algoritmos específicos que aprovechen la nueva estructura del problema proporcionen mejores soluciones.

2.2. Modelo de programación matemática

Sean los grafos completos $G^1 = (V^1, A^1)$ y $G^2 = (V^2, A^2)$ que representan, respectivamente, las localizaciones donde se debe recoger y entregar la mercancía. Para cada $\delta \in \{1, 2\}$ el grafo G^δ tiene unos pesos c_{ij}^δ asociados a cada arista (i, j) que representan la distancia o coste de viaje del vértice i al j. Si n es el número de clientes, los conjuntos de vértices son $V^\delta = \{0, 1, \cdots, n\}$, $\delta \in \{1, 2\}$, donde 0 representa el depósito central y el resto, $1, \ldots, n$, se asocian a los n clientes. Se dispone de m pilas para almacenar la mercancía, todas con la misma capacidad Q, y se supone que $mQ \geq n$ para que siempre exista solución factible.

Es útil definir los conjuntos $V_*^\delta = V^\delta \backslash \{0\}$ y $P = \{1, \cdots, m\}$ para representar los vértices asociados a los clientes en cada grafo $\delta \in \{1, 2\}$ y las pilas disponibles, respectivamente. Así, se considera un conjunto de encargos $D = \{1, \cdots, n\}$, de manera que la mercancía correspondiente a cada encargo $i \in D$ debe ser recogida en el vértice $i \in V_*^1$ del grafo G^1, almacenada en una determinada pila $p \in P$ y entregada en el vértice $i \in V_*^2$ del grafo G^2.

2.2.1. Variables del modelo

A continuación se presentan los distintos conjuntos de variables utilizados para modelizar el problema. Si no se indica lo contrario se supone que $\delta \in \{1, 2\}$.

Variables para la construcción de rutas

El conjunto de variables $\{x_{ij}^\delta\}$ indica cómo diseñar las rutas a seguir en los grafos de entrega y recogida.

$$x_{ij}^\delta = \begin{cases} 1 & \text{si } j \text{ se visita inmediatamente después que } i \text{ en el grafo } \delta \\ 0 & \text{en otro caso} \end{cases} \quad \forall i, j \in V^\delta,\ i \neq j$$

Variables de precedencia entre los vértices

El conjunto de variables $\{y_{ij}^\delta\}$ indica la precedencia entre los vértices de cada grafo. Estas variables se utilizan para imponer la regla LIFO que rige las pilas.

$$y_{ij}^\delta = \begin{cases} 1 & \text{si } i \text{ se visita antes que } j \text{ en el grafo } \delta \\ 0 & \text{en otro caso} \end{cases} \quad \forall i, j \in V_*^\delta,\ i \neq j$$

Variables de asignación de pilas

El conjunto de variables $\{z_{ip}\}$ asigna una pila a cada encargo.

$$z_{ip} = \begin{cases} 1 & \text{si la mercancía del encargo } i \text{ se almacena en la pila } p \\ 0 & \text{en otro caso} \end{cases} \quad \forall i \in D, \forall p \in P$$

2.2.2. Función objetivo

El coste de una solución es la suma de las distancias de las rutas recorridas en ambos grafos, y su expresión es:

$$\min \sum_{i,j \in V^1, i \neq j} c_{ij}^1 \cdot x_{ij}^1 + \sum_{i,j \in V^2, i \neq j} c_{ij}^2 \cdot x_{ij}^2$$

Las variables $\{x_{ij}^\delta\}$ son las únicas que se ven implicadas en la función objetivo por ser las que describen las rutas a recorrer, los otros conjuntos de variables se utilizan exclusivamente para asegurar la factibilidad de la solución.

2.2.3. Restricciones

Las restricciones del modelo se presentan a continuación.

Restricciones de conservación de flujo

Las restricciones (2.1) y (2.2) son las denominadas *restricciones de conservación de flujo*, que imponen que el vehículo visite cada vértice una y sólo una vez en cada grafo. De esta manera, una unidad de flujo entra y sale en cada vértice.

$$\sum_{i \in V^\delta} x_{ij}^\delta = 1 \qquad \forall \delta, \forall j \in V^\delta \tag{2.1}$$

$$\sum_{j \in V^\delta} x_{ij}^\delta = 1 \qquad \forall \delta, \forall i \in V^\delta \tag{2.2}$$

Restricciones para la definición de las variables de precedencia

Las restricciones (2.3) y (2.4) garantizan la definición correcta de las variables de precedencia $\{y_{ij}^\delta\}$. Para cada pareja de vértices i, j y cada grafo δ, o bien i se visita antes que j, o bien j se visita antes que i. Esta condición se expresa en las restricciones (2.3):

$$y_{ij}^\delta + y_{ji}^\delta = 1 \qquad \forall \delta, \forall i, j \in V_*^\delta, i \neq j \tag{2.3}$$

El conjunto de restricciones (2.4) se ocupa de imponer la relación de transitividad entre las variables de precedencia: si el cliente i se visita antes que el k y el k antes que el j, necesariamente el cliente i se debe visitar antes que el j.

$$y_{ik}^{\delta} + y_{kj}^{\delta} \leq y_{ij}^{\delta} + 1 \qquad \forall \delta, \forall i, j, k \in V_*^{\delta}, i \neq j \neq k, i \neq k \tag{2.4}$$

Si en el grafo δ se visita el vértice j inmediatamente después del i será $x_{ij}^{\delta} = 1$, y el vértice i debe preceder al vértice j en la ruta correspondiente al grafo δ, esto es, $y_{ji}^{\delta} = 1$, de donde resultan las restricciones (2.5). Nótese que estas restricciones no eliminan ninguna solución factible.

$$x_{ij}^{\delta} \leq y_{ij}^{\delta} \qquad \forall \delta, \forall i, j \in V_*^{\delta}, i \neq j \tag{2.5}$$

Restricciones LIFO en las pilas

Las restricciones (2.6) imponen el cumplimiento de la regla de descarga LIFO en cada pila: si los encargos i y j están asignados a la misma pila y la mercancía correspondiente al encargo i se recoge antes que la correspondiente al j, entonces la mercancía del encargo j debe ser entregada *antes* que la del i.

$$y_{ij}^{1} + z_{ip} + z_{jp} \leq 3 - y_{ij}^{2} \qquad \forall p \in P, \forall i, j \in D, i \neq j \tag{2.6}$$

Restricciones de asignación de pilas

Los conjuntos de restricciones (2.7) y (2.8) aseguran, respectivamente, que cada encargo se asigne a una pila y sólo a una y que no se exceda la capacidad máxima de las pilas.

$$\sum_{p \in P} z_{ip} = 1 \qquad \forall i \in D \tag{2.7}$$

$$\sum_{i \in D} z_{ip} \leq Q \qquad \forall p \in P \tag{2.8}$$

Restricciones de variables 0/1

Las restricciones (2.9) indican que todas las variables del modelo son binarias.

$$x, y, z \in \{0, 1\} \tag{2.9}$$

Observación 1. El número de variables binarias x_{ij}^{δ}, y_{ij}^{δ}, z_{ip} utilizadas es, respectivamente,

$2(n+1)^2$, $2n^2$, mn, por lo que el número total de variables es $4n^2+(m+4)n+1$. El número de restricciones de cada tipo, siguiendo el orden en que se presentan, es respectivamente $2(n+1)$, $2(n+1)$, $2\binom{n}{2}$, $2n(n-1)(n-2)$, $2n(n-1)$, $n(n-1)m$, n, m, y así el número total de restricciones es $4+m+(6-m)n+(m-3)n^2+2n^3$.

Nótese que las condiciones de eliminación de subciclos (*Subcycle Elimination Constraints*, SEC) en las rutas de recogida y entrega están definidas implícitamente en el modelo propuesto, como se establece a continuación en el teorema 1.

Teorema 1. *Las restricciones de conservación de flujo, dadas en* (2.1) *y* (2.2)*, junto con las restricciones que definen las relaciones de precedencia, dadas en* (2.3)*,* (2.4) *y* (2.5)*, evitan la formación de subciclos en las rutas del problema.*

Demostración

Supongamos, sin pérdida de generalidad, que los vértices $1,2,\cdots,k \in V_*^\delta$ forman un subciclo en el grafo δ, de manera que el orden en que se visitan es $1,2,\cdots,k-1,k,k+1=1$, etc. En lo sucesivo se omitirá el superíndice δ en las variables x_{ij} e y_j, por simplicidad. Así, se tiene que $x_{12}=x_{23}=\cdots=x_{k-1,k}=1$ y $x_{k,1}=1$ para cerrar el subciclo, es decir,

$$x_{j,j+1}=1 \quad \forall j \in \{1,\cdots,k\}=A. \tag{2.10}$$

Entonces, (2.5) y (2.10) $\Rightarrow \quad y_{j,j+1} \geq x_{j,j+1}=1 \ \forall j \in A \ \Rightarrow \quad y_{j,j+1}=1 \ \forall j \in A.$

Por la propiedad transitiva de la relación de precedencia entre clientes, dada en (2.4), se tiene lo siguiente:

$$\begin{array}{lll} y_{12}=1, \quad y_{23}=1 & \Rightarrow & y_{13}=1 \\ y_{13}=1, \quad y_{34}=1 & \Rightarrow & y_{14}=1 \\ & \vdots & \\ y_{1,k-1}=1, \quad y_{k-1,k}=1 & \Rightarrow & y_{1,k}=1 \end{array}$$

Pero también $x_{k,1}=1 \ \Rightarrow \ y_{k,1}=1$, lo cual contradice (2.3), ya que $y_{1,k}+y_{k,1}=1+1=2$.

Por tanto, los vértices $1,2,\cdots,k$ no pueden formar un subciclo en el grafo δ. □

Observación 2. Las restricciones (2.1)-(2.5) aseguran la correcta definición de las variables de ruta x_{ij}^δ y de precedencia y_{ij}^δ sin eliminar ninguna ruta factible. Para cada $\delta \in \{1,2\}$, fijadas las variables de ruta x_{ij}^δ sólo existe un conjunto de variables de precedencia y_{ij}^δ que verifican (2.1)-(2.5), y de la misma forma fijadas las variables de ruta y_{ij}^δ sólo existe un conjunto de variables de precedencia x_{ij}^δ que verifican (2.1)-(2.5). Así las variables x_{ij}^δ y y_{ij}^δ dan la misma información, pero las primeras facilitan el cálculo del coste de la solución y las segundas la imposición de las restricciones de carga y precedencia, además de impedir la formación de subciclos.

Así, el modelo del DTSPMS inicialmente propuesto, Modelo 1, se presenta a continuación.

Modelo 1. DTSPMS

$$\min \sum_{i,j \in V^1, i \neq j} c_{ij}^1 \cdot x_{ij}^1 + \sum_{i,j \in V^2, i \neq j} c_{ij}^2 \cdot x_{ij}^2 \tag{2.11}$$

sujeto a

$$\sum_{i \in V^\delta, i \neq j} x_{ij}^\delta = 1 \qquad \forall \delta, \forall j \in V^\delta \tag{2.12}$$

$$\sum_{j \in V^\delta, j \neq i} x_{ij}^\delta = 1 \qquad \forall \delta, \forall i \in V^\delta \tag{2.13}$$

$$y_{ij}^\delta + y_{ji}^\delta = 1 \qquad \forall \delta, \forall i, j \in V_*^\delta, i \neq j \tag{2.14}$$

$$y_{ik}^\delta + y_{kj}^\delta \leq y_{ij}^\delta + 1 \qquad \forall \delta, \forall i, j, k \in V_*^\delta, i \neq j \neq k, i \neq k \tag{2.15}$$

$$x_{ij}^\delta \leq y_{ij}^\delta \qquad \forall \delta, \forall i, j \in V_*^\delta, i \neq j \tag{2.16}$$

$$y_{ij}^1 + z_{ip} + z_{jp} \leq 3 - y_{ij}^2 \qquad \forall p \in P, \forall i, j \in D, i \neq j \tag{2.17}$$

$$\sum_{p \in P} z_{ip} = 1 \qquad \forall i \in D \tag{2.18}$$

$$\sum_{i \in D} z_{ip} \leq Q \qquad \forall p \in P \tag{2.19}$$

$$x, y, z \in \{0, 1\} \tag{2.20}$$

2.3. Problemas relacionados

El DTSPMS es un problema de rutas de vehículos con entrega y recogida de mercancías que se caracteriza por la inclusión de ciertas restricciones de precedencia y capacidad adicionales inducidas por la utilización de varias pilas en el contenedor del vehículo. Ha sido poco tratado por el momento en la literatura especializada, pero sin embargo guarda una estrecha relación con otros problemas de rutas de vehículos con determinadas restricciones adicionales que sí han sido estudiados con mayor detalle. En las siguientes secciones se presentan algunos de estos problemas, junto con sus modelos de programación matemática, destacando sus similitudes con el DTSPMS.

2.3.1. Problema del viajante con recogida y entrega de mercancías (TSPPD)

En el *Traveling Salesman Problem with Pickup and Delivery* (TSPPD) se dispone de un único vehículo que tiene que satisfacer los encargos de una serie de clientes, los cuales requieren la recogida de cierta mercancía en un determinado lugar de origen (*pickup*) y la posterior entrega en el correspondiente destino (*delivery*). El vehículo de que se dispone tiene una capacidad de carga limitada, siendo el objetivo del problema encontrar un ciclo hamiltoniano de coste mínimo que cumpla con todas las recogidas y entregas programadas respetando las restricciones de precedencia entre las localizaciones origen y destino y no excediendo la capacidad máxima del vehículo.

Uno de los primeros en abordar el TSPPD fue Stein (1978), que propuso un algoritmo heurístico para el caso de un vehículo sin límite de capacidad. Le siguieron otros autores como Psaraftis (1983a,b), Kubo y Kasugai (1990) y Van der Bruggen et al. (1993), que estudiaron la versión del problema que trata el transporte de personas (*dial a ride problems*) e incluye ventanas de tiempo. Mosheiov (1994) y Anily y Mosheiov (1994) también estudiaron el TSPPD, refiriéndose a él como el *Traveling Salesman Problem with Delivery and Backhauls*. Más recientemente, Gendreau, Laporte y Vigo (1999) propusieron dos algoritmos heurísticos para el TSPPD, una metaheurística de búsqueda tabú y una heurística basada en el algoritmo de Christofides (1976) para el TSP, y Baldacci et al. (2003) diseñaron un algoritmo exacto de tipo Branch & Bound. Süral y Bookbinder (2003) abordaron un problema similar al TSPPD en el que algunos clientes de recogida se pueden visitar de forma opcional, aportando un cierto beneficio si su mercancía es recogida, y Alshamrani et al. (2007) estudiaron la versión periódica del problema.

Sean $D = \{1, \cdots, n\}$ el conjunto de encargos que deben atenderse, $V^1 = \{1, \cdots, n\}$ el conjunto de localizaciones de recogida, $V^2 = \{n+1, \cdots, 2n\}$ el conjunto de localizaciones de entrega y $V^* = V^1 \cup V^2$. Así, el problema se define sobre un grafo dirigido completo $G = (V, A)$, donde $V = \{0, 2n+1\} \cup V^1 \cup V^2$ es el conjunto de vértices del problema, 0 representa el origen del que parte el vehículo, $2n+1$ representa el depósito destino (que puede coincidir con el origen) y A es el conjunto de arcos. Cada arco $(i, j) \in A$ tiene asociado un coste $c_{ij} \geq 0$ relacionado con la distancia que separa i de j o el coste necesario para viajar de i a j.

Cada encargo $i \in D$ está asociado a la localización de recogida $i \in V^1$ y a la de entrega $i + n \in V^2$, y tiene una demanda asociada d_i que representa la cantidad de mercancía que hay que recoger en i y entregar en $i + n$. Para cada $i \in V^1$ la demanda d_i también puede asociarse directamente a la localización de recogida i, representando la cantidad de mercancía a recoger en i, y por extensión se define la demanda de cada localización de entrega $i + n \in V^2$ como $d_{i+n} = -d_i$, representando la cantidad de mercancía a entregar en $i + n$ con signo negativo. Por último se definen las demandas $d_0 = d_{2n+1} = 0$ asociadas a los depósitos inicial y final y la capacidad máxima del vehículo, que se denotará por Q.

El conjunto de variables binarias $\{x_{ij}\}$ permite reconstruir la ruta a seguir.

$$x_{ij} = \begin{cases} 1 & \text{si el vértice } j \text{ se visita inmediatamente después del } i \\ 0 & \text{en otro caso} \end{cases} \quad \forall i, j \in V$$

Para conseguir una formulación lineal del problema es necesario introducir otro conjunto de variables continuas $\{f_{ij}\}$ que representen la carga del vehículo cuando éste recorre cada arco $(i, j) \in A$.

Sea $\mathcal{V} = \{H \subseteq V \mid 0 \in H,\ 2n+1 \notin H,\ \exists i \in D \text{ tal que } i \notin H,\ n+i \in H\} \subseteq 2^V$ la colección formada por los subconjuntos de vértices que contienen al depósito inicial y no al final y que contienen la localización de entrega de algún encargo pero no la correspondiente localización de recogida.

Así, el Modelo 2 que se presenta a continuación es una formulación lineal del TSPPD:

Modelo 2. TSPPD

$$\min \sum_{i \in V} \sum_{j \in V} c_{ij} x_{ij} \tag{2.21}$$

sujeto a

$$\sum_{j \in V, j \neq i} x_{ij} = 1 \qquad \forall i \in V^* \tag{2.22}$$

$$\sum_{i \in V, i \neq j} x_{ij} = 1 \qquad \forall j \in V^* \tag{2.23}$$

$$\sum_{j \in V} x_{0j} = 1 \tag{2.24}$$

$$\sum_{i \in V} x_{i,2n+1} = 1 \tag{2.25}$$

$$\sum_{i,j \in H} x_{ij} \leq |H| - 1 \qquad \forall H \subseteq V^*,\ |S| \geq 2 \tag{2.26}$$

$$\sum_{i,j \in H} x_{ij} \leq |H| - 2 \qquad \forall H \in \mathcal{V} \tag{2.27}$$

$$\sum_{j \in V} f_{ij} - \sum_{j \in V} f_{ji} = d_i \qquad \forall i \in V^* \tag{2.28}$$

$$\sum_{j \in V^1} f_{0j} = 0 \tag{2.29}$$

$$\sum_{i \in V^2} f_{i,2n+1} = 0 \tag{2.30}$$

$$0 \leq f_{ij} \leq Q x_{ij} \qquad \forall i, j \in V \tag{2.31}$$

$$x_{ij} \in \{0, 1\} \qquad \forall i, j \in V \tag{2.32}$$

Las restricciones (2.22)-(2.25) aseguran que se visiten todas las localizaciones del problema una y sólo una vez, incluyendo los depósitos origen y destino. Las restricciones (2.26) garantizan la eliminación de subciclos y las restricciones (2.27) imponen las relaciones de precedencia entre cada pareja de localizaciones i e $i+n$. En (2.28)-(2.30) se definen correctamente las variables de carga, que controlan que no se exceda la capacidad máxima del vehículo manteniéndose entre las cotas definidas en (2.31). Por último en (2.32) se indica que las variables x_{ij} son binarias.

2.3.2. Problema del viajante con recogida y entrega de mercancías y restricciones de carga LIFO (TSPPDL)

El *Traveling Salesman Problem with Pickup and Delivery and LIFO Loading* (TSPPDL) es un TSPPD al que se le han añadido restricciones de precedencia LIFO que controlan el orden en que se almacena y extrae la mercancía del vehículo. El problema surge en situaciones reales de transporte de mercancías en las que sólo se puede acceder a la carga del vehículo por una puerta y dicha carga no se puede reorganizar, de manera que en cada momento sólo se puede disponer de la mercancía recogida en último lugar. Así, la mercancía recogida se almacena formando una pila que sigue un principio LIFO y sólo se puede entregar en cada momento la mercancía situada en la cabecera de la pila.

El TSPPDL fue estudiado por primera vez por Ladany y Mehrez (1984) y Kalantari et al. (1985). También ha sido abordado por Fischetti y Toth (1989), Savelsbergh (1990), Healy y Moll (1995), Pacheco Bonrostro (1997a,b), Ruland y Rodin (1997), Renaud, Boctor y Ouenniche (2000) y Renaud, Boctor y Laporte (2002). Cassani (2004) y Cassani y Righini (2004) propusieron una heurística greedy y un algoritmo de búsqueda en entorno variable descendente con cuatro estructuras de entornos distintas, y posteriormente Carrabs et al. (2007) añadieron otras 3 estructuras de entornos nuevas y propusieron un algoritmo de búsqueda en entorno variable general que utiliza las 7 estructuras de entornos. En Cordeau et al. (2006) puede encontrarse un artículo de recopilación reciente sobre el TSPPDL.

En el Modelo 2 del TSPPD sólo se impone que cada localización $i \in V^1$ donde se recoge mercancía se visite antes que la correspondiente localización $i + n$ donde debe entregarse (restricciones (2.27)). En un TSPPDL es necesario imponer también que la localización

$i+n$ sólo puede visitarse cuando toda la mercancía recogida en las localizaciones visitadas después que i ya haya sido entregada, es decir, cuando la mercancía de la localización $j+n$ ha sido ya entregada para cada $j \in V^1$ visitada después que i. Esta condición se consigue añadiendo al Modelo 2 el siguiente conjunto de restricciones:

$$\sum_{j \in V} f_{ji} - \sum_{j \in V} f_{n+i,j} = 0 \qquad \forall i \in V^1 \tag{2.33}$$

las cuales imponen que la carga del vehículo justo antes de guardar la mercancía de cada localización de recogida i sea idéntica a la carga del vehículo justo después de dejar la mercancía en la correspondiente localización de entrega $i+n$.

2.3.3. Problema del viajante con múltiples pilas (TSPMS)

El *Traveling Salesman Problem with Multiple Stacks* (TSPMS) es un TSPPD en el que la carga se puede almacenar en el único vehículo disponible formando varias pilas, cada una de las cuales se rige por un principio LIFO. Las recogidas y entregas se realizan en la misma región, debiendo ser alternadas para no exceder la capacidad máxima del vehículo y para respetar las restricciones de precedencia impuestas por las pilas, las cuales pueden considerarse como sistemas LIFO independientes.

En la literatura sólo se ha encontrado una referencia a un problema similar al TSPMS, denominado 1-PDMS (*1-Pickup and Delivery with Multiple Stacks*), en el que la cantidad de carga a recoger y entregar puede ser diferente para cada cliente. En este trabajo (Côté et al., 2009) se presenta una heurística basada en una Búsqueda en Entornos Grandes Adaptativa (ALNS: *Adaptative Large Neighborhood Search*) que se aplica a la resolución de instancias del 1-PDMS de distintos tamaños.

El TSPMS es un problema que generaliza el TSPPDL y el DTSPMS. El TSPPDL es un TSPMS con una única pila, mientras que el DTSPMS es un TSPMS en el que se exige que primero se recoja la mercancía de todos los encargos y a continuación se entregue en las localizaciones correspondientes. Este problema es, obviamente, más complicado de resolver que el TSPPD, el TSPPDL y el DTSPMS. Su dificultad ya se atisba en su modelización como problema de programación matemática, que es parecida a la del DTSPMS pero requiere la utilización de 5 grupos de variables, uno de ellos con 3 índices.

Sea $P = \{1, \cdots, m\}$ el conjunto de pilas disponibles, las cuales tienen una capacidad máxima Q. El resto de la notación utilizada en el modelo que se presenta a continuación es la misma que para el TSPPD, incluyendo demandas d_j para cada $j \in V^*$, que serán positivas para las localizaciones de recogida y negativas para las de entrega.

Variables para la construcción de la ruta

El conjunto de variables $\{x_{ij}\}$ indica cómo diseñar la ruta a seguir.

$$x_{ij} = \begin{cases} 1 & \text{si el vértice } j \text{ se visita inmediatamente después del } i \\ 0 & \text{en otro caso} \end{cases} \quad \forall i, j \in V$$

Variables de precedencia

El conjunto de variables $\{y_{ij}\}$ indica la precedencia entre los vértices del grafo.

$$y_{ij} = \begin{cases} 1 & \text{si el vértice } i \text{ se visita antes que el } j \\ 0 & \text{en otro caso} \end{cases} \quad \forall i, j \in V^*,\ i \neq j$$

Variables de asignación de pilas

El conjunto de variables $\{z_i^p\}$ indica cómo se realiza la asignación de cada encargo a una pila.

$$z_i^p = \begin{cases} 1 & \text{si la mercancía asociada al encargo } i \text{ se almacena en la pila } p \\ 0 & \text{en otro caso} \end{cases} \quad \forall i \in D, \forall p \in P$$

Variables de orden en las pilas

Diremos que $i \in V^1$ opera en la pila $p \in P$ si la mercancía recogida en i se almacena en p; por otra parte, diremos que $i \in V^2$ opera en la pila $p \in P$ si la mercancía que se entrega en i se saca de la pila p. El conjunto de variables $\{t_{ij}^p\}$ indica el orden que siguen los encargos asignados a cada pila.

$$t_{ij}^p = \begin{cases} 1 & \text{si } j \text{ opera inmediatamente después que } i \text{ en la pila } p \\ 0 & \text{en otro caso} \end{cases} \quad \forall i, j \in V^*, \forall p \in P$$

Variables de carga

El conjunto de variables $\{q_i^p\}$ indica el nivel de carga de las pilas en cada instante del recorrido.

$q_i^p \equiv$ carga de la pila p inmediatamente después de visitar la localización $i, \forall i \in V^*, \forall p \in P$.

El Modelo 3 que se presenta a continuación es una formulación válida para el problema.

Modelo 3. TSPMS

$$\min \sum_{i\in V}\sum_{j\in V, j\neq i} c_{ij}x_{ij} \tag{2.34}$$

sujeto a

$$\sum_{j\in V, j\neq i} x_{ij} = 1 \qquad \forall i \in V^* \tag{2.35}$$

$$\sum_{i\in V, i\neq j} x_{ij} = 1 \qquad \forall j \in V^* \tag{2.36}$$

$$\sum_{j\in V} x_{0j} = 1 \tag{2.37}$$

$$\sum_{i\in V} x_{i,2n+1} = 1 \tag{2.38}$$

$$y_{ij} + y_{ji} = 1 \qquad \forall i,j \in V^*,\ i \neq j \tag{2.39}$$

$$y_{ik} + y_{kj} \leq y_{ij} + 1 \qquad \forall i,j,k \in V^*,\ i \neq j \neq k,\ i \neq k \tag{2.40}$$

$$x_{ij} \leq y_{ij} \qquad \forall i,j \in V^*,\ i \neq j \tag{2.41}$$

$$y_{i,i+n} = 1 \qquad \forall i \in V^1 \tag{2.42}$$

$$\sum_{p\in P} z_i^p = 1 \qquad \forall i \in D \tag{2.43}$$

$$y_{ij} + z_i^p + z_j^p \leq 3 - y_{i+n,j+n} \qquad \forall p \in P, \forall i,j \in V^1,\ i \neq j \tag{2.44}$$

$$t_{ij}^p \leq y_{ij},\ t_{ij}^p \leq z_i^p,\ t_{ij}^p \leq z_j^p \qquad \forall p \in P, \forall i,j \in V^*,\ i \neq j \tag{2.45}$$

$$t_{j,i+n}^p = 0 \qquad \forall p \in P, \forall i,j \in V^1,\ i \neq j \tag{2.46}$$

$$t_{ij}^p \leq 3 - z_k^p - y_{ik} - y_{kj} \qquad \forall i,j,k \in V^*,\ i \neq j \neq k,\ i \neq k \tag{2.47}$$

$$\sum_{p\in P}\sum_{i\in V^*}\sum_{j\in V^*} t_{ij}^p = 2n - m \tag{2.48}$$

$$q_i^p \geq d_i z_i^p \qquad \forall i \in V^1 \tag{2.49}$$

$$q_j^p - q_i^p - d_j \geq (-Q - d_j)(1 - t_{ij}^p) \qquad \forall p \in P, \forall i,j \in V^*,\ i \neq j \tag{2.50}$$

$$q_j^p - q_i^p - d_j \leq (Q - d_j)(1 - t_{ij}^p) \qquad \forall p \in P, \forall i,j \in V^*,\ i \neq j \tag{2.51}$$

$$0 \leq q_i^p \leq Q \qquad \forall i \in V^* \tag{2.52}$$

$$x_{ij}, y_{ij}, z_i^p, t_{ij}^p \in \{0,1\} \qquad \forall p \in P, \forall i,j \in V, i \neq j \tag{2.53}$$

Las restricciones de conservación de flujo, que se encargan de que cada vértice sea visitado una y sólo una vez, vienen dadas en (2.35)-(2.38). Las restricciones (2.39)-(2.42) y (2.43) aseguran, respectivamente, que las variables de precedencia $\{y_{ij}\}$ y las de asignación de pilas $\{z_i^p\}$ se definan correctamente. El principio LIFO que debe seguir cada pila por separado queda impuesto por el conjunto de restricciones dado en (2.44). Las restricciones

(2.45)-(2.48) definen adecuadamente las variables de orden en las pilas $\{t_{ij}^p\}$, asegurando la compatibilidad con las variables de precedencia y asignación de pilas. La restricción (2.48) obliga a cualquier solución factible del problema a utilizar todas las pilas disponibles, con el fin de poder calcular de forma exacta el número de variables t_{ij}^p que tienen valor 1, que es igual al número de localizaciones menos el número de pilas utilizadas. Las variables de carga q_i^p de las pilas se definen en (2.49)-(2.51) y se acotan en (2.52) para evitar que las pilas excedan su capacidad máxima Q. Por último, en (2.53) se indica que todas las variables utilizadas son binarias, salvo las variables de carga que son continuas.

Las restricciones (2.50) y (2.51) linealizan la siguiente restricción:

$$t_{ij}^p = 1 \;\Rightarrow\; q_j^p = q_i^p + d_j \tag{2.54}$$

que significa que si j opera inmediatamente después que i en la pila p, la cantidad de mercancía almacenada en la pila p tras abandonar la localización j es la que había al abandonar la localización i más la demanda asociada a j, d_j. Dicha demanda d_j puede ser positiva, aumentando la carga de la pila p, si j es una localización de recogida, o negativa, disminuyendo la carga de la pila p, si es una localización de entrega.

2.3.4. Problema de Rutas de Vehículos con Múltiples Pilas (MP-VRP)

El *Multi-Pile Vehicle Routing Problem* (MP-VRP) fue introducido recientemente en Doerner et al. (2007) y viene motivado por la problemática existente en empresas madereras dedicadas al transporte de productos para la construcción. La empresa en cuestión debe decidir cómo almacenar los productos madereros en los vehículos disponibles y posteriormente qué rutas deben seguir para realizar los repartos. Se trata de un problema de optimización combinatoria que combina problemas de empaquetamiento, rutas y secuenciación y que guarda una estrecha relación con el DTSPMS.

Los productos pueden ser de distintos tipos y dimensiones. Los del mismo tipo se agrupan en palés, formando *ítems*. Se dispone de una flota de vehículos idénticos, en los que los ítems pueden almacenarse formando varias pilas con una determinada capacidad máxima y con disciplina LIFO a las que se puede acceder de forma independiente. En primer lugar hay que decidir cómo almacenar todos los ítems en los distintos vehículos de forma factible, y posteriormente diseñar las rutas para los vehículos de manera que los productos que se entregan se puedan extraer de la parte superior de las pilas sin realizar ningún tipo de reorganización de la carga.

El MP-VRP, al igual que el DTSPMS, ha sido aún poco tratado en la literatura. Doerner et al. (2007) introducen el problema y le aplican dos algoritmos heurísticos basados en la Búsqueda Tabú (Glover, 1989) y mecanismos de Colonias de Hormigas (Dorigo et al., 1996). Se considera un subproblema del MP-VRP denominado *One Vehicle Loading*

Problem (1-VPL), que consiste en determinar si existe alguna forma factible de cargar un conjunto dado de ítems en un solo vehículo de manera que una ruta dada sea factible; se proponen heurísticas sencillas para resolver dicho subproblema, las cuales son utilizadas como subrutinas en los algoritmos propuestos para resolver el MP-VRP. En Tricoire et al. (2008) se presenta un algoritmo de Búsqueda en Entorno Variable para resolver el problema de forma heurística y un algoritmo Branch & Bound para resolver el problema de forma exacta. Además se propone un algoritmo de programación dinámica para resolver de forma exacta el 1-VPL, que sigue siendo un subproblema importante en los procedimientos de resolución propuestos.

El modelo de programación matemática para el MP-VRP que se presenta a continuación, introducido en Tricoire et al. (2008), es una adaptación de la formulación clásica del problema de rutas de vehículos como problema de flujo. Sea V el conjunto de clientes a servir (incluyendo el depósito, denotado por 0), E el conjunto de aristas formadas por pares de elementos de V y c_e el coste de viaje asociado a la arista $e \in E$. Para cada $S \subseteq V$, el conjunto de aristas con un extremo en S se denota por $\delta(S)$ y el conjunto de aristas con ambos extremos en S se denota por $E(S)$; además, $k(S)$ representa una cota inferior para el número mínimo de vehículos necesarios para servir a los clientes de S, y si los clientes de S no se pueden servir utilizando un único vehículo se dice que S es *1-VLP infactible*. Para simplificar la notación, $\delta(\{i\})$ se denota también como $\delta(i)$. Por último, una secuencia ordenada de clientes $P = (r_1, \cdots, r_m)$ (excluyendo el depósito) se denomina *camino*, y el conjunto de aristas pertenecientes a un camino P se denota por $A(P) = \{(r_1, r_2), \cdots, (r_{m-1}, r_m)\}$.

El modelo propuesto para el MP-VRP (Modelo 4) es el siguiente:

Modelo 4. MP-VRP

$$\min \sum_{e \in E} c_e x_e \tag{2.55}$$

sujeto a

$$\sum_{e \in \delta(i)} x_e = 2 \qquad \forall i \in V \tag{2.56}$$

$$\sum_{e \in E(S)} x_e \leq |S| - k(S) \qquad \forall S \subseteq V, |S| \geq 2 \tag{2.57}$$

$$\sum_{e \in A(P)} x_e \leq |A(P)| - 1 \qquad \forall P \subseteq V : P \text{ es 1-VLP infactible} \tag{2.58}$$

$$x_e \in \{0, 1\} \qquad \forall e \in E \backslash \delta(0) \tag{2.59}$$

$$x_e \in \{0, 1, 2\} \qquad \forall e \in \delta(0) \tag{2.60}$$

Las restricciones (2.56) imponen que cada cliente sea visitado exactamente una vez, las restricciones generalizadas de eliminación de subciclos (2.57) imponen la conectividad de las rutas y las restricciones (2.58) aseguran que sólo se acepten rutas que sean factibles con respecto a las restricciones de carga. Por último, los conjuntos de restricciones (2.59) y (2.60) indican que todas las variables de decisión x_e son binarias salvo las asociadas a aristas e con un único extremo en el depósito, las cuales corresponden a rutas con un solo cliente y pueden también tomar valor 2.

Capítulo 3

Complejidad y obtención de cotas para el DTSPMS

El DTSPMS, descrito con detalle en el capítulo anterior, es un problema NP-duro que combina características de los problemas de rutas de vehículos con la inclusión de restricciones de precedencia y de carga. En la Sección 3.1, presentada a continuación, se estudia con detalle la complejidad del problema y de algunos subproblemas interesantes. Posteriormente, en la Sección 3.2 se proponen algunas estrategias para la obtención de cotas inferiores para el DTSPMS, basadas en la Relajación Lagrangiana, la Descomposición de Dantzig-Wolfe y la generación de Planos de Corte.

3.1. Complejidad del DTSPMS

El DTSPMS es una generalización del Problema del Viajante (TSP), ya que cualquier instancia del TSP se puede transformar en una instancia equivalente del DTSPMS en la que la matriz de distancias de la red de recogida es la misma que la del TSP inicial y la distancia entre cualquier par de vértices en la red de entrega es la misma; así, la ruta de recogida de cualquier solución óptima del DTSPMS es una solución óptima del TSP. Por tanto, el DTSPMS es un problema NP-duro y entra dentro del conjunto de problemas de optimización complicados para los que no se conocen algoritmos de resolución en tiempo polinómico. En Garey y Johnson (1979) se expone detalladamente la teoría de la NP-completitud y se realiza un estudio extensivo de la complejidad de algoritmos y problemas de optimización.

En realidad el DTSPMS es un problema notablemente más complicado de resolver que el TSP debido a las restricciones de precedencia impuestas por las condiciones de carga del vehículo, que enlazan las rutas de entrega y recogida del problema y dificultan

enormemente la caracterización de las soluciones. Se han conseguido resolver instancias del TSP con miles de vértices de forma exacta en un tiempo de cómputo razonable, mientras que las instancias más grandes del DTSPMS que se han podido resolver hasta el momento de forma exacta (Petersen et al., 2009) tienen sólo 20 encargos y 4 pilas de capacidad 5, lo cual pone de manifiesto la dificultad del problema.

El DTSPMS puede descomponerse de forma natural en tres partes perfectamente diferenciadas, aunque estrechamente relacionadas: la ruta de recogida, la ruta de entrega y la asignación de pilas. Resolviendo de forma independiente los TSPs asociados a las regiones de recogida y entrega se pueden obtener rutas de recogida y entrega que sean óptimas (o de muy buena calidad) consideradas aisladamente; sin embargo, es muy poco probable que para una pareja de estas rutas exista una asignación de pilas compatible y por tanto puedan formar una solución del DTSPMS. En una de las pruebas computacionales realizadas, para una instancia del DTSPMS con 33 encargos y 3 pilas de capacidad 11 se generaron 10.000 parejas de rutas de recogida y entrega 2-óptimas, y para ninguna de ellas existía una asignación de pilas factible que permitiese la obtención de una solución del DTSPMS. Esto pone de manifiesto que los dos TSPs no pueden considerarse aisladamente, sino que deben resolverse de forma conjunta a través de la asignación de pilas.

En Casazza et al. (2009) se analiza la complejidad de algunos subproblemas de un DTSPMS con pilas sin límite de capacidad. Se determina que los siguientes subproblemas de dicha simplificación sin límite de capacidad se pueden resolver en tiempo polinómico:

1. Dadas una ruta de recogida y una de entrega, encontrar una asignación de pilas compatible con el mínimo número de pilas.
2. Dadas una ruta de recogida y una de entrega, encontrar una asignación de pilas parcial compatible que incluya el mayor número posible de encargos utilizando a lo sumo m pilas.
3. Dadas una ruta de recogida y una asignación de pilas, encontrar la ruta de entrega compatible de longitud mínima.
4. Dadas una ruta de entrega y una asignación de pilas, encontrar la ruta de recogida compatible de longitud mínima.

En la literatura no existen resultados similares para el DTSPMS original con a lo sumo Q encargos por pila, pero aunque los resultados anteriores no son válidos para dicho problema, algunos de estos subproblemas sin capacidades sí pueden resultar útiles para la obtención de cotas inferiores del problema original.

En la siguiente sección se proponen algunas estrategias para la obtención de cotas inferiores para el DTSPMS. Debido a la gran complejidad del problema, dichas estrategias

y los enfoques exactos que se pueden derivar de ellas sólo pueden aplicarse a instancias de pequeño tamaño, por lo que para resolver instancias de tamaño realista es necesaria la utilización de heurísticas. Por esta razón los capítulos 4, 5 y 6 se dedican en su totalidad al diseño de algoritmos heurísticos que permitan la obtención de soluciones de buena calidad en instancias de tamaño realista y al desarrollo de las herramientas necesarias para ellos.

3.2. Obtención de cotas inferiores

A partir de la relajación lineal del Modelo 1 se pueden obtener cotas inferiores del DTSPMS de forma sencilla, aunque normalmente dichas cotas estarán muy alejadas de la solución óptima del problema original. Otra forma de obtener cotas inferiores del problema es considerar la relajación del Modelo 1 que se obtiene eliminando las restricciones (2.17)-(2.19), que se reduce a dos TSPs independientes correspondientes a las rutas de recogida y entrega. A partir de esta nueva relajación, para cuya resolución es necesario resolver dos TSPs con n vértices, se obtienen cotas inferiores que normalmente son mejores que las dadas por la relajación lineal pero que suelen estar todavía muy alejadas de la solución óptima. En las siguientes secciones se propondrán algunas estrategias alternativas para la obtención de cotas inferiores para el DTSPMS basadas en la Relajación Lagrangiana, la Descomposición de Dantzig-Wolfe y la Generación de planos de corte.

3.2.1. Relajación Lagrangiana

En esta sección se propone un modelo de Relajación Lagrangiana para el DTSPMS, para cuya definición es necesario añadir un conjunto nuevo de variables para representar los *conflictos* entre los encargos de una solución.

Definición 14. Fijadas las rutas de recogida y entrega para una instancia del DTSPMS, dos encargos se podrán asignar a la misma pila si se recogen y entregan en orden inverso. Así, decimos que existe un *conflicto* entre dos encargos i y j si dichos encargos no se pueden asignar a la misma pila, lo cual ocurre cuando dichos encargos se recogen y entregan en el mismo orden.

Para representar los conflictos entre las rutas de recogida y entrega introducimos un nuevo conjunto de variables w_{ij} definidas como

$$w_{ij} = \begin{cases} 1 & \text{si el encargo } i \text{ se recoge y entrega antes que el } j \\ 0 & \text{en otro caso} \end{cases} \quad \forall i, j \in D,\ i \neq j$$

de manera que los encargos i y j están en conflicto si y sólo si $w_{ij} = 1$ ó $w_{ji} = 1$.

Añadiendo las variables $w_{ij}, \forall i,j \in D,\ i \neq j$ al Modelo 1 y sustituyendo las restricciones (2.17), (2.20) por (3.1)-(3.3), se obtiene otra formulación del DTSPMS, descrita con detalle en el Modelo 5, en la que la relación entre las rutas de la solución y la asignación de pilas se expresa de manera alternativa. Las restricciones (3.1) imponen que si i se recoge y entrega antes que j entonces $w_{ij} = 1$, mientras que las restricciones (3.2) aseguran que si $w_{ij} = 1$ los encargos i y j no se asignan a la misma pila.

$$y^1_{ij} + y^2_{ij} \leq 1 + w_{ij} \qquad \forall i,j \in D, i \neq j \tag{3.1}$$

$$z_{ip} + z_{jp} \leq 2 - w_{ij} \qquad \forall p \in P, \forall i,j \in D, i \neq j \tag{3.2}$$

$$x, y, z, w \in \{0,1\} \tag{3.3}$$

Modelo 5. DTSPMS con conflictos

$$\min \sum_{i,j \in V^1, i \neq j} c^1_{ij} \cdot x^1_{ij} + \sum_{i,j \in V^2, i \neq j} c^2_{ij} \cdot x^2_{ij} \tag{3.4}$$

sujeto a

$$\sum_{i \in V^1, i \neq j} x^1_{ij} = 1 \qquad \forall j \in V^1 \tag{3.5}$$

$$\sum_{i \in V^2, i \neq j} x^2_{ij} = 1 \qquad \forall j \in V^2 \tag{3.6}$$

$$\sum_{j \in V^1, j \neq i} x^1_{ij} = 1 \qquad \forall i \in V^1 \tag{3.7}$$

$$\sum_{j \in V^2, j \neq i} x^2_{ij} = 1 \qquad \forall i \in V^2 \tag{3.8}$$

$$x^1_{ij} \leq y^1_{ij} \qquad \forall i,j \in V^1_*, i \neq j \tag{3.9}$$

$$x^2_{ij} \leq y^2_{ij} \qquad \forall i,j \in V^2_*, i \neq j \tag{3.10}$$

$$y^1_{ij} + y^1_{ji} = 1 \qquad \forall i,j \in V^1_*, i \neq j \tag{3.11}$$

$$y^2_{ij} + y^2_{ji} = 1 \qquad \forall i,j \in V^2_*, i \neq j \tag{3.12}$$

$$y^1_{ik} + y^1_{kj} \leq y^1_{ij} + 1 \qquad \forall i,j,k \in V^1_*, i \neq j \neq k, i \neq k \tag{3.13}$$

$$y^2_{ik} + y^2_{kj} \leq y^2_{ij} + 1 \qquad \forall i,j,k \in V^2_*, i \neq j \neq k, i \neq k \tag{3.14}$$

$$y^1_{ij} + y^2_{ij} \leq 1 + w_{ij} \qquad \forall i,j \in D, i \neq j \tag{3.15}$$

$$z_{ip} + z_{jp} \leq 2 - w_{ij} \qquad \forall p \in P, \forall i,j \in D, i \neq j \tag{3.16}$$

$$\sum_{p \in P} z_{ip} = 1 \qquad \forall i \in D \tag{3.17}$$

$$\sum_{i \in D} z_{ip} \leq Q \qquad \forall p \in P \tag{3.18}$$

$$x, y, z, w \in \{0,1\} \tag{3.19}$$

La introducción de las variables w_{ij} en el Modelo 5 permite descomponer el problema inicial en cinco subproblemas más sencillos obtenidos a partir de la relajación lagrangiana de las restricciones (3.9), (3.10) y (3.15). Los multiplicadores lagrangianos asociados a las restricciones (3.9), (3.10) y (3.15) se denotarán, respectivamente, por $\mu_{ij}^1 \geq 0$, $\mu_{ij}^2 \geq 0$ y $\lambda_{ij} \geq 0$, y la función objetivo obtenida tras la relajación lagrangiana de (3.9), (3.10) y (3.15) será

$$z_{LR} = \sum_{(i,j)\in A^1} (c_{ij}^1 + \mu_{ij}^1)x_{ij}^1 + \sum_{(i,j)\in A^2} (c_{ij}^2 + \mu_{ij}^2)x_{ij}^2 + \\ + \sum_{i,j=1}^{n} (\lambda_{ij} - \mu_{ij}^1)y_{ij}^1 + \sum_{i,j=1}^{n} (\lambda_{ij} - \mu_{ij}^2)y_{ij}^2 - \sum_{i,j=1}^{n} \lambda_{ij} w_{ij} \tag{3.20}$$

donde en todos los sumatorios se supone que $i \neq j$.

Observación 3. Los multiplicadores μ_{ij}^δ sólo están definidos para $i, j = 1, \cdots, n$, pero en los dos primeros sumatorios de z_{LR} los índices i y j pueden tomar el valor 0 (valor asociado al depósito). Por ello, para que la notación sea consistente, definimos $\mu_{0j}^\delta = 0 = \mu_{i0}^\delta = 0$ para cada $i, j = 1, \cdots, n$, $\delta = 1, 2$.

El problema de optimización obtenido a partir de la relajación lagrangiana de las restricciones (3.9), (3.10) y (3.15) del Modelo 5 es

$$LR : \min\{z_{LR} \text{ sujeto a } (3.5) - (3.8), (3.11) - (3.14), (3.16) - (3.19)\}$$

Gracias a la eliminación de las restricciones (3.9) y (3.10), que relacionan las variables x_{ij}^δ e y_{ij}^δ, y las restricciones (3.15), que relacionan las variables y_{ij}^δ y w_{ij}, el problema relajado LR puede descomponerse en cinco subproblemas independientes:

$$TSP_1' : \min\left\{ z_{TSP}^{1'} = \sum_{(i,j)\in A^1} (c_{ij}^1 + \mu_{ij}^1)x_{ij}^1 \text{ , sujeto a (3.5), (3.6), } x_{ij}^1 \in \{0,1\} \right\}$$

$$TSP_2' : \min\left\{ z_{TSP}^{2'} = \sum_{(i,j)\in A^2} (c_{ij}^2 + \mu_{ij}^2)x_{ij}^2 \text{ , sujeto a (3.7), (3.8), } x_{ij}^2 \in \{0,1\} \right\}$$

$$LOP_1 : \min\left\{ z_{LOP}^1 = \sum_{i,j=1}^{n} (\lambda_{ij} - \mu_{ij}^1)y_{ij}^1 \text{ , sujeto a (3.11), (3.13), } y_{ij}^1 \in \{0,1\} \right\}$$

$$LOP_2 : \min\left\{ z^2_{LOP} = \sum_{i,j=1}^{n} (\lambda_{ij} - \mu^2_{ij}) y^2_{ij} \text{ , sujeto a (3.12), (3.14), } y^2_{ij} \in \{0,1\} \right\}$$

$$COL : \min\left\{ z_{COL} = -\sum_{i,j=1}^{n} \lambda_{ij} w_{ij} \text{ , sujeto a (3.16) } - \text{ (3.18), } z_{ip}, w_{ij} \in \{0,1\} \right\}$$

Los problemas LOP_1 y LOP_2 consisten en encontrar una permutación de los n encargos, representada por las variables de precedencia y^δ_{ij}, de mínimo coste, y por tanto son Problemas de Ordenación Lineal (conocidos por sus siglas en inglés LOP: *Linear Ordering Problems*). El LOP es un problema NP-duro que ha sido extensamente estudiado en la literatura. En Schiavinotto y Stützle (2004) y Charon y Hudry (2007) se puede encontrar información detallada sobre el LOP y multitud de referencias adicionales.

El problema COL consiste en particionar los n encargos dados en m conjuntos con a lo sumo Q elementos de manera que la suma de los costes $-\lambda_{ij}$ para cada pareja de encargos (i,j) asignados al mismo conjunto de la partición sea mínima, por lo que se trata de un problema de Coloración de Grafos con particiones con tamaño máximo. Este tipo de problemas también han sido extensamente estudiados en la literatura (Kubale, 2004) y son NP-duros en general.

Las restricciones de eliminación de subciclos se imponen implícitamente en el modelo 5 por la acción de las variables de precedencia sobre las de ruta, pero al desconectar ambos grupos de variables estas restricciones implícitas se pierden en los subproblemas TSP'_1 y TSP'_2, que no son TSPs sino simples problemas de asignación. Para que dichos subproblemas mantengan la estructura original y sean TSPs se añaden las SEC explícitamente, obteniendo los nuevos subproblemas siguientes:

$$TSP_1 : \min\left\{ z^1_{TSP} = \sum_{(i,j)\in A^1} (c^1_{ij} + \mu^1_{ij}) x^1_{ij} \text{ sujeto a (3.5), (3.6)}, SEC_1, \; x^1_{ij} \in \{0,1\} \right\}$$

$$TSP_2 : \min\left\{ z^2_{TSP} = \sum_{(i,j)\in A^2} (c^2_{ij} + \mu^2_{ij}) x^2_{ij} \text{ sujeto a (3.7), (3.8)}, SEC_2, \; x^2_{ij} \in \{0,1\} \right\}$$

donde $SEC_\delta = \{h^\delta_i - h^\delta_j + (n+1)x^\delta_{ij} \leq n, \; \forall i,j \in V^\delta_*, \; i \neq j, \; h^\delta_i \in \mathbb{R}, \; \forall i \in V^\delta_*\}$ para cada $\delta \in \{1,2\}$.

Para cada conjunto de multiplicadores $\mu_{ij}^1 \geq 0$, $\mu_{ij}^2 \geq 0$, $\lambda_{ij} \geq 0$ el valor $z_{LB} = z_{TSP}^1 + z_{TSP}^2 + z_{LOP}^1 + z_{LOP}^2 + z_{COL}$ obtenido sumando los valores óptimos de los subproblemas TSP_1, TSP_2, LOP_1, LOP_2, COL, es una cota inferior del DTSPMS.

El problema Dual Lagrangiano pretende calcular la mejor cota que se puede obtener a partir de la correspondiente Relajación Lagrangiana, encontrando el mejor conjunto posible de multiplicadores. Existen distintas técnicas para resolver el Dual Lagrangiano, como por ejemplo el algoritmo del subgradiente (Fisher, 1985). Dicho algoritmo ha sido aplicado a la resolución del Dual Lagrangiano correspondiente a la Relajación Lagrangiana propuesta para el DTSPMS, pero en ninguna de las instancias consideradas se ha conseguido mejorar las cotas inferiores obtenidas con todos los multiplicadores igualados a 0 (que coinciden con las cotas inferiores obtenidas resolviendo un TSP asociado a cada grafo de forma independiente).

Tras no obtener resultados satisfactorios con el algoritmo del subgradiente decidimos aplicar otra técnica diferente basada en la Reformulación de Dantzig-Wolfe para resolver el problema Dual Lagrangiano. Si las restricciones relajadas en la Relajación Lagrangiana coinciden con las restricciones de acople de la descomposición de Dantzig-Wolfe, los valores óptimos del problema Dual Lagrangiano y de la relajación lineal de la Reformulación de Dantzig-Wolfe coinciden (Geoffrion, 1974, Fisher, 1981), y los valores óptimos de las variables duales asociadas a las restricciones de acople del problema maestro de la Reformulación de Dantzig-Wolfe forman un conjunto óptimo de multiplicadores para las restricciones relajadas en la Relajación Lagrangiana (Magnanti et al., 1976). La relajación lineal de la Reformulación de Dantzig-Wolfe se puede resolver mediante un proceso de generación de columnas en el cual los subproblemas que se deben resolver para generar las nuevas columnas (con coste reducido negativo) a añadir al problema maestro son los mismos que se obtienen en la Relajación Lagrangiana (TSP_1, TSP_2, LOP_1, LOP_2, COL), salvo por una constante en la función objetivo. En la sección siguiente se detalla cómo construir la Reformulación de Dantzig-Wolfe del Modelo 5 asociada a la Relajación Lagrangiana propuesta.

3.2.2. Descomposición de Dantzig-Wolfe. Generación de columnas

Para evaluar la efectividad de la Relajación Lagrangiana propuesta consideramos la descomposición de Dantzig-Wolfe del Modelo 5, cuya relajación lineal puede resolverse mediante generación de columnas. El Modelo 5 se puede formular de forma equivalente como el Modelo 6, que se presenta a continuación.

Modelo 6. DTSPMS con conflictos. Formulación convexificada.

$$\min \sum_{i,j\in V^1, i\neq j} c^1_{ij}\cdot x^1_{ij} + \sum_{i,j\in V^2, i\neq j} c^2_{ij}\cdot x^2_{ij} \tag{3.21}$$

sujeto a

$$(x^1_{ij}) \in \text{conv}\{(3.5),(3.7), SEC_1, 0\leq x^1_{ij}\leq 1\} \tag{3.22}$$
$$(x^2_{ij}) \in \text{conv}\{(3.6),(3.8), SEC_2, 0\leq x^2_{ij}\leq 1\} \tag{3.23}$$
$$x^1_{ij}\leq y^1_{ij} \quad \forall i,j\in V^1_*, i\neq j \tag{3.24}$$
$$x^2_{ij}\leq y^2_{ij} \quad \forall i,j\in V^2_*, i\neq j \tag{3.25}$$
$$(y^1_{ij}) \in \text{conv}\{(3.11),(3.13), 0\leq y^1_{ij}\leq 1\} \tag{3.26}$$
$$(y^2_{ij}) \in \text{conv}\{(3.12),(3.14), 0\leq y^2_{ij}\leq 1\} \tag{3.27}$$
$$y^1_{ij}+y^2_{ij}\leq 1+w_{ij} \quad \forall i,j\in D, i\neq j \tag{3.28}$$
$$\big((z_{ip}),(w_{ij})\big) \in \text{conv}\{(3.16),(3.17),(3.18), 0\leq w_{ij}\leq 1, 0\leq z_{ip}\leq 1\} \tag{3.29}$$

donde (x^δ_{ij}), (y^δ_{ij}), (z_{ip}) y (w_{ij}) representan los vectores de variables $(x^\delta_{00},\ldots,x^\delta_{nn})$, $(y^\delta_{11},\ldots,y^\delta_{nn})$, $(z_{11},\ldots,z_{nm})$ y $(w_{11},\ldots,w_{nn})$, respectivamente.

Sean Ω_1 y Ω_2 los conjuntos de puntos extremos (enteros) de las envolturas convexas de (3.22) y (3.23), respectivamente, y representemos por $(\bar{x}^1_{ija})$ y $(\bar{x}^2_{ija})$ los puntos de cada una de ellas. Análogamente, sea Γ_1 el conjunto de puntos extremos $(\bar{y}^1_{ijb})$ de la envoltura convexa de (3.26), Γ_2 el conjunto de puntos extremos $(\bar{y}^2_{ijb})$ de la envoltura convexa de (3.27) y Λ el conjunto de puntos extremos $\big((\bar{z}_{ipc}),(\bar{w}_{ijc})\big)$ de la envoltura convexa de (3.29). La descomposición de Dantzig-Wolfe del Modelo 6 con restricciones de acople (3.24), (3.25), (3.28), que eran las restricciones relajadas en la Relajación Lagrangiana, descompone la matriz de restricciones en cinco bloques, cada uno de los cuales se corresponde a uno de los subproblemas de la Relajación Lagrangiana. La formulación de dicha descomposición, con columnas $(\bar{x}^1_{ija})$, $(\bar{x}^2_{ija})$, $(\bar{y}^1_{ijb})$, $(\bar{y}^2_{ijb})$ y $(\bar{w}_{ijc})$, se presenta en el Modelo 7.

Modelo 7. Descomposición de Dantzig-Wolfe

$$\min v_{MP} = \sum_{a\in\Omega_1}\Big(\sum_{i,j\in A_1} c^1_{ij}\bar{x}^1_{ija}\Big)z_a + \sum_{a\in\Omega_2}\Big(\sum_{i,j\in A_2} c^2_{ij}\bar{x}^2_{ija}\Big)z_a \tag{3.30}$$

sujeto a

$$\sum_{a\in\Omega_1}\bar{x}^1_{ija}z_a \leq \sum_{b\in\Gamma_1}\bar{y}^1_{ijb}r_b \qquad \forall i,j\in V^1_*, i\neq j \tag{3.31}$$

$$\sum_{a\in\Omega_2} \bar{x}^2_{ija} z_a \leq \sum_{b\in\Gamma_2} \bar{y}^2_{ijb} r_b \qquad \forall i,j \in V^2_*, i \neq j \tag{3.32}$$

$$\sum_{b\in\Gamma_1} \bar{y}^1_{ijb} r_b + \sum_{b\in\Gamma_2} \bar{y}^2_{ijb} r_b \leq 1 + \sum_{c\in\Lambda} \bar{w}_{ijc} t_c \qquad \forall i,j \in D, i \neq j \tag{3.33}$$

$$\sum_{a\in\Omega_1} z_a = \sum_{a\in\Omega_2} z_a = \sum_{b\in\Gamma_1} r_b = \sum_{b\in\Gamma_2} r_b = \sum_{c\in\Lambda} t_c = 1 \tag{3.34}$$

$$0 \leq z,\, r,\, t \leq 1 \tag{3.35}$$

La variable z_a vale 1 si se selecciona la ruta a y 0 si no, r_b vale 1 si se selecciona el grupo de precedencias b, 0 si no, y t_c vale 1 si se selecciona una coloración c, 0 si no. Los problemas de generar puntos extremos de Ω_1 y Ω_2 son análogos a TSP_1 y TSP_2, respectivamente, mientras que los problemas de generar puntos extremos de Γ_1 y Γ_2 son análogos a LOP_1 y LOP_2 y el problema de generar puntos extremos de Λ es análogo a COL.

En todos los casos considerados en las pruebas computacionales realizadas el valor óptimo de la descomposición de Dantzig-Wolfe es igual a la cota inicial obtenida resolviendo dos TSPs independientes, indicando que, en efecto, dicha cota era la mejor que se podía obtener a partir de la Relajación Lagrangiana. También se realizaron pruebas similares relajando sólo uno de los conjuntos de restricciones anteriormente elegidos con la intención de incrementar las cotas inferiores obtenidas, pero los resultados obtenidos fueron de nuevo los mismos.

Por tanto, concluimos que ni la Relajación Lagrangiana inicialmente propuesta ni la Reformulación de Dantzig-Wolfe asociada han producido resultados satisfactorios en la obtención de buenas cotas inferiores para el DTSPMS.

3.2.3. Planos de Corte

En las dos secciones anteriores quedó de manifiesto la dificultad de obtener buenas cotas inferiores para el DTSPMS utilizando Relajación Lagrangiana y resolviendo mediante métodos de Generación de Columnas la descomposición de Dantzig-Wolfe asociada, ya que con ninguna de estas estrategias fue posible incrementar la cota inferior obtenida eliminando directamente las restricciones de precedencia. En esta sección se propone otra estrategia diferente a las anteriores, basada en la adición de planos de corte al problema relajado donde se han eliminado las restricciones de precedencia, que ha resultado más eficaz en la práctica.

Definición 15. Consideremos la formulación del DTSPMS dada por el Modelo 5. El Modelo 8 que se presenta a continuación es una formulación de la *Relajación por Precedencias* del DTSPMS (RP), que es la relajación que se obtiene eliminando las restricciones de precedencia del Modelo 5.

Modelo 8. Relajación por Precedencias del DTSPMS (RP)

$$\min \sum_{i,j\in V^1, i\neq j} c^1_{ij} \cdot x^1_{ij} + \sum_{i,j\in V^2, i\neq j} c^2_{ij} \cdot x^2_{ij} \tag{3.36}$$

sujeto a

$$\sum_{i\in V^1, i\neq j} x^1_{ij} = 1 \qquad \forall j \in V^1 \tag{3.37}$$

$$\sum_{i\in V^2, i\neq j} x^2_{ij} = 1 \qquad \forall j \in V^2 \tag{3.38}$$

$$\sum_{j\in V^1, j\neq i} x^1_{ij} = 1 \qquad \forall i \in V^1 \tag{3.39}$$

$$\sum_{j\in V^2, j\neq i} x^2_{ij} = 1 \qquad \forall i \in V^2 \tag{3.40}$$

$$x^1_{ij} \leq y^1_{ij} \qquad \forall i,j \in V^1_*, i \neq j \tag{3.41}$$

$$x^2_{ij} \leq y^2_{ij} \qquad \forall i,j \in V^2_*, i \neq j \tag{3.42}$$

$$y^1_{ij} + y^1_{ji} = 1 \qquad \forall i,j \in V^1_*, i \neq j \tag{3.43}$$

$$y^2_{ij} + y^2_{ji} = 1 \qquad \forall i,j \in V^2_*, i \neq j \tag{3.44}$$

$$y^1_{ik} + y^1_{kj} \leq y^1_{ij} + 1 \qquad \forall i,j,k \in V^1_*, i \neq j \neq k, i \neq k \tag{3.45}$$

$$y^2_{ik} + y^2_{kj} \leq y^2_{ij} + 1 \qquad \forall i,j,k \in V^2_*, i \neq j \neq k, i \neq k \tag{3.46}$$

$$x, y \in \{0,1\} \tag{3.47}$$

Observación 4. El Modelo 8 es separable y se reduce a dos TSPs independientes, por lo que su solución óptima son dos ciclos hamiltonianos r_1 (recogida) y r_2 (entrega) asociados, respectivamente, a los conjuntos de variables x^1_{ij} y x^2_{ij}. Las variables de precedencia y^δ_{ij} no son necesarias para su resolución y podrían eliminarse, sustituyéndose por un conjunto de restricciones de eliminación de subciclos (SEC), pero se mantienen en esta formulación porque serán útiles más adelante a la hora de añadir planos de corte.

El procedimiento que se propone para la obtención de cotas inferiores para el DTSPMS se basa en la aplicación del Algoritmo de Planos de Corte (Algoritmo 1) al caso en el que el problema P es el DTSPMS y la relajación Q es la Relajación por Precedencias del DTSPMS. Es necesario diseñar un algoritmo de separación que permita identificar los planos de corte a introducir en la relajación, para lo cual se utilizará el concepto de conflicto (definición 14), así como algunas propiedades del llamado grafo de conflictos, que se introduce a continuación.

Definición 16. Sean r_1 y r_2 una ruta de recogida y otra de entrega para una instancia del DTSPMS, respectivamente. El *grafo de conflictos* determinado por r_1 y r_2 se define como

$G^{r_2}_{r_1} = (D, A^{r_2}_{r_1})$, donde el conjunto de vértices $D = \{1, \cdots, n\}$ es el conjunto de encargos y la arista (i, j) pertenece a $A^{r_2}_{r_1}$ si y sólo si existe un conflicto entre i y j.

Fijadas las rutas de recogida y entrega, el problema de encontrar una asignación de pilas compatible con el mínimo número de pilas (sin capacidades) es equivalente al problema de encontrar una coloración del grafo de conflictos con el mínimo número de colores (Casazza et al., 2009), y el problema de encontrar el mayor conjunto de encargos con conflictos dos a dos es equivalente al problema de encontrar un clique de orden máximo en el grafo de conflictos. Además, el mínimo número de colores necesarios coincide con el orden del clique de orden máximo.

Así, si hay m pilas disponibles y el grafo de conflictos contiene un clique $\{u_1, \cdots, u_l\}$ con $l > m$, todos estos encargos deberán ser asignados a pilas diferentes y por tanto las rutas de recogida y entrega fijadas no pueden formar una solución factible del DTSPMS. Los planos de corte a añadir deben impedir que el clique de orden máximo identificado se vuelva a formar al resolver la nueva relajación, permitiendo la obtención de una nueva solución óptima. Si el cardinal del clique de orden máximo es menor o igual que m esto significa que los encargos se pueden distribuir en las m pilas disponibles y por tanto no se puede identificar ningún plano de corte más con esta estrategia; en este caso dos situaciones son posibles: que la solución sea factible, y por tanto se tenga la solución óptima, o que cualquier asignación de pilas compatible con la solución dada viole las restricciones de capacidad, y por tanto sólo se disponga de una cota inferior.

Definición 17. Un grafo no dirigido $G = (V, A)$ con $V = \{1, \cdots, n\}$ se dice que es un *grafo permutación* si existe una permutación $\sigma \in BIY(V, V)$ tal que $(i, j) \in A$ con $i < j$ si y sólo si $\sigma(i) < \sigma(j)$.

La resolución de las Relajaciones por Precedencias en cada iteración es compleja, ya que se trata de TSPs con restricciones adicionales (véase por ejemplo Escudero y Ortuño, 1997 y Alonso et al., 2003), que son los planos de corte añadidos, pero la identificación de los cliques de orden máximo es sencilla, ya que el grafo de conflictos es un grafo permutación (proposición 1) y en este tipo de grafos la identificación de un clique de orden máximo se puede realizar en tiempo polinomial (proposición 2).

Proposición 1. *Para cualquier pareja de rutas de recogida y entrega r_1 y r_2, el grafo de conflictos $G^{r_2}_{r_1} = (D, A^{r_2}_{r_1})$ es un grafo permutación.*

Demostración

Sin pérdida de generalidad y renombrando los encargos del problema si es necesario supongamos que $r_1 \equiv [1, 2, \cdots, n]$ y $r_2 \equiv [u_1, u_2, \cdots, u_n]$. Entonces para cada $i < j$, $(u_i, u_j) \in A^{r_2}_{r_1}$ si y sólo si $u_i < u_j$, por lo que $G^{r_2}_{r_1}$ es el grafo permutación dado por r_2. □

Proposición 2. *El problema de encontrar un clique de orden máximo en un grafo permutación con n vértices se puede resolver en tiempo polinomial, $O(n \log n)$.*

Demostración

Huang y Wang (2007). □

Huang y Wang (2007) proponen un algoritmo para obtener un clique de orden máximo en un grafo permutación en tiempo $O(n \log n)$, que es el algoritmo que utilizaremos para encontrar un clique de orden máximo en el grafo de conflictos.

Proposición 3. *Sean r_1 y r_2 rutas de recogida y entrega del DTSPMS y sea $G_{r_1}^{r_2}$ el grafo de conflictos. Si $U = \{u_1, \cdots, u_l\}$ es un clique de $G_{r_1}^{r_2}$ con $l > m$ y u_i se recoge antes que u_j si $i < j$, la desigualdad*

$$y^1_{u_1,u_2} + \cdots + y^1_{u_{l-1},u_l} \; + \; y^2_{u_1,u_2} + \cdots + y^2_{u_{l-1},u_l} \leq 2l - 3 \tag{3.48}$$

deducida a partir de U es válida para el DTSPMS y es violada por la pareja de rutas r_1 y r_2.

Demostración

En ninguna solución factible del DTSPMS puede haber $l > m$ encargos que se visiten y recojan en el mismo orden, y por tanto alguna de las variables de precedencia $y^1_{u_1,u_2}, \cdots, y^1_{u_{l-1},u_l}$, $y^2_{u_1,u_2}, \cdots, y^2_{u_{l-1},u_l}$ será distinta de 1, haciendo que el lado derecho de la restricción (3.48) sea a lo sumo $2(l-1) - 1 = 2l - 3$. Por tanto, la restricción (3.48) es válida para el DTSPMS.

El orden en que se recogen los encargos de U es $(u_1, \cdots, u_l)$, pero como forman un clique en el grafo de conflictos el orden de entrega será el mismo. Entonces las variables de precedencia asociadas a las rutas r_1 y r_2 verificarán que $y^1_{u_1,u_2} = \cdots = y^1_{u_{l-1},u_l} = 1$ y $y^2_{u_1,u_2} = \cdots = y^2_{u_{l-1},u_l} = 1$, por lo que el lado derecho de la restricción (3.48) sería $2(l-1) > 2l - 3$, produciéndose la violación de dicha restricción. □

La proposición 3 establece que, si la solución óptima de la relajación por Precedencias del DTSPMS es la asociada a las rutas r_1 y r_2, entonces la desigualdad (3.48), si existe, es un plano de corte.

Las proposiciones 1 y 2 establecen cómo identificar un clique de orden máximo en el grafo de conflictos y la proposición 3 cómo generar a partir de dicho clique un plano de corte que añadir a la relajación considerada. Aunque ese único plano de corte es suficiente para descartar la pareja de rutas inicial, añadiendo en el mismo paso otros planos de

corte inducidos por otros cliques de orden mayor que m se puede reducir el número de iteraciones del algoritmo de planos de corte, reduciendo el número de relajaciones a resolver y consiguiendo una convergencia más rápida.

Por ejemplo, si $l > m+1$, para cada subconjunto de cardinal $m+1$ $\{v_1, \cdots, v_{m+1}\} \subset U = \{u_1, \cdots, u_l\}$ del clique de orden máximo U identificado por el algoritmo, la desigualdad

$$y^1_{v_1 v_2} + \cdots + y^1_{v_m v_{m+1}} + y^2_{v_1 v_2} + \cdots + y^2_{v_m v_{m+1}} \leq 2m - 1 \tag{3.49}$$

es válida para el DTSPMS y más fuerte que la desigualdad (3.48) inducida por el clique U. Por tanto es más conveniente añadir todos estos planos de corte inducidos por subconjuntos de U de cardinal $m+1$ que el único plano de corte, más débil, inducido directamente por U.

También es posible que, además del clique U identificado por el algoritmo, existan otros cliques de orden máximo. Para encontrar algunos de ellos se puede proceder eliminando de uno en uno los encargos de U y aplicando el algoritmo de Huang y Wang sobre las nuevas rutas con $n-1$ encargos. Si el nuevo clique de orden máximo tiene $l-1$ encargos se descarta, pero si tiene l entonces es un clique de orden máximo distinto de U, ya que al menos difiere en un encargo. De cada uno de los nuevos cliques de orden l se pueden obtener nuevos planos de corte.

A continuación se presenta el pseudocódigo del *Algoritmo de Separación* propuesto para la inclusión de planos de corte en la Relajación por Precedencias del DTSPMS. Los parámetros de entrada del algoritmo son las dos rutas r_1 y r_2 que forman la solución óptima de la Relajación del DTSPMS considerada y el parámetro de salida es un conjunto de planos de corte válidos Ψ. Si dicho conjunto es vacío el algoritmo de planos de corte para, ya que no se puede encontrar ningún plano de corte adicional.

Algoritmo 11. Algoritmo de Separación

1. Construir el grafo de conflictos $G^{r_2}_{r_1} = (D, A^{r_2}_{r_1})$ y encontrar un clique $U = \{u_1, \cdots, u_l\}$ de orden máximo mediante el algoritmo de Huang y Wang (2007). Poner $\Psi = \emptyset$.

2. Si $l \leq m$, FIN. En otro caso, para cada subconjunto $\{v_1, \cdots, v_{m+1}\} \subset U$ añadir a Ψ el plano de corte definido en (3.49).

3. Para cada $u \in U$, aplicar de nuevo el algoritmo de Huang y Wang (2007) al grafo de conflictos obtenido eliminando el encargo u del grafo $G^{r_2}_{r_1}$, obteniendo un clique de orden máximo $W = \{w_1, \cdots, w_r\}$. Si $r > l$ y $W \nsubseteq U$, añadir a Ψ la desigualdad de la forma (3.48) asociada al clique W.

Llamaremos **Algoritmo de Cortes con Conflictos** (CCA, *Conflict Cuts Algorithm*) al Algoritmo de Planos de Corte (Algoritmo 1) aplicado al DTSPMS con la Relajación por Precedencias y el Algoritmo de Separación (Algoritmo 11) que acaba de ser introducido. Con dicho algoritmo se han podido mejorar algunas cotas inferiores para instancias con 12 encargos y 2 pilas de capacidad 6 y para instancias de 18 encargos y 3 pilas de capacidad 6, que no han sido resueltas de forma óptima en la literatura. Un resumen de los resultados computacionales correspondientes a estas instancias se presentarán más adelante en el Capítulo 7.

Debido a la gran complejidad del DTSPMS, las estrategias para la obtención de cotas inferiores y los enfoques exactos propuestos hasta el momento han sido poco efectivos y sólo han podido aplicarse a instancias de pequeño tamaño, por lo que para resolver instancias de tamaño realista es necesaria la utilización de heurísticas. Por esta razón los capítulos siguientes (4, 5 y 6) se dedican al diseño de algoritmos heurísticos y al desarrollo de las herramientas necesarias para ellos. Las heurísticas no garantizan la obtención de soluciones óptimas, pero son capaces de ofrecer soluciones factibles de buena calidad para instancias de tamaños grandes utilizando poco tiempo de cómputo.

Capítulo 4

Tipología de soluciones en el DTSPMS. Estructuras de entornos

En este capítulo se estudia con detalle como manejar de forma adecuada las soluciones del DTSPMS dentro del ámbito de las heurísticas. En la Sección 4.1 se discute cómo se pueden representar las soluciones del problema sin utilizar variables binarias, permitiendo un manejo más sencillo de ellas en los algoritmos heurísticos. En la Sección 4.2 se proponen varios métodos para generar soluciones del DTSPMS de distintos tipos, las cuales se pueden utilizar como soluciones iniciales en cualquier algoritmo heurístico. En la Sección 4.3 se presentan diversas estructuras de entornos para soluciones factibles, las cuales permiten la realización de distintos movimientos que definen búsquedas locales basadas en estrategias diferentes. Por último, en la Sección 4.4 se presentan dos estructuras de entornos adicionales para soluciones potenciales, las cuales permiten diseñar de forma sencilla algoritmos heurísticos que utilizan soluciones intermedias no factibles en el proceso de búsqueda.

4.1. Una representación operativa para las soluciones

La representación de soluciones del DTSPMS a través de variables binarias introducida en la Sección 2.2 es adecuada para plantear el modelo de programación lineal entera (Modelo 1), pero no resulta útil en la práctica para programar algoritmos heurísticos ni para explicar de forma sencilla cómo dichos algoritmos modifican la solución de partida para obtener otras mejores. Por ello, es necesario introducir otra representación más operativa que permita manejar los elementos de una solución del problema más fácilmente, como la que se presenta a continuación.

Definición 18. Una instancia del DTSPMS viene determinada por el número de encargos n que se deben atender, las matrices de costes o distancias $C_1 = (c_{ij}^1)$ y $C_2 = (c_{ij}^2)$ y

el número m de pilas disponibles junto con su capacidad máxima Q. Se denotará por $I = \{1, \cdots, n\}$ el conjunto de posiciones posibles en una ruta, $D = \{1, \cdots, n\}$ el conjunto de encargos del problema y $P = \{1, \cdots, m\}$ el conjunto de pilas disponibles. Una solución factible del DTSPMS queda definida por una ruta de recogida, una ruta de entrega y una asignación de pilas.

Las rutas de recogida (ruta 1) y entrega (ruta 2) vienen definidas, respectivamente, por dos permutaciones π_1 y π_2 del conjunto D de la forma

$$\begin{array}{rcll} \pi_\delta : I & \longrightarrow & D, & \delta \in \{1,2\} \\ i & \longmapsto & \pi_\delta(i) \equiv \text{cliente } i\text{-ésimo} & \end{array}$$

de manera que $\pi_\delta(i)$, $i \in I$, $\delta \in \{1,2\}$ representa el encargo asociado al cliente que ocupa el lugar i-ésimo en la ruta δ. Es importante observar que, al ser permutaciones, las aplicaciones π_1 y π_2 son biyectivas.

La asignación de pilas viene dada por una aplicación

$$\begin{array}{rcl} \lambda : D & \longrightarrow & P \\ u & \longmapsto & \lambda(u) \equiv \text{pila asignada al encargo } u \end{array}$$

que asigna a cada encargo la pila en la que debe ser almacenado. La aplicación λ no es inyectiva, pero sí será sobreyectiva siempre y cuando $n > (m-1)Q$, es decir, siempre que no exista ninguna asignación de pilas factible en la que alguna de las pilas no se utilice.

Observación 5. Nótese que una asignación de pilas λ indica sólo en qué pila se almacena cada encargo, pero por sí misma no determina su posición dentro de ella. El orden en que los encargos son almacenados en cada pila viene dado por una de las dos rutas de la solución, es decir, por el orden de recogida o entrega de los encargos.

Dados tres conjuntos finitos A, B y C, A y B con el mismo cardinal, denotamos por $\mathrm{Apl}(A,C) = \{\lambda : A \longrightarrow C \mid \lambda \text{ es una aplicación}\}$ al conjunto de aplicaciones de A a C y por $\mathrm{Biy}(A,B) = \{\pi : A \longrightarrow B \mid \pi \text{ es biyectiva}\}$ al conjunto de aplicaciones biyectivas de A a B.

Con la notación anterior, cualquier solución del problema queda determinada por tres aplicaciones $\pi_1, \pi_2 \in \mathrm{Biy}(I,D)$ y $\lambda \in \mathrm{Apl}(D,P)$. Sin embargo, es obvio que una terna (π_1, π_2, λ) puede representar una solución no factible del problema, ya que no está garantizado que se respeten las restricciones de precedencia y capacidad máxima impuestas por las pilas.

Proposición 4. *Sean $\pi_1, \pi_2 \in Biy(I,D)$ y $\lambda \in Apl(D,P)$. La terna (π_1, π_2, λ) representa una solución factible del* DTSPMS si y sólo si *verifica las siguientes condiciones:*

$$| \lambda^{-1}(k) | \leq Q \qquad \forall k \in P \tag{4.1}$$

$$\pi_1^{-1}(i) < \pi_1^{-1}(j) \Leftrightarrow \pi_2^{-1}(i) > \pi_2^{-1}(j) \qquad \forall i, j \in D, i \neq j, \text{ tales que } \lambda(i) = \lambda(j) \tag{4.2}$$

Demostración

$\Leftarrow$) Sea (π_1, π_2, λ) verificando (4.1) y (4.2). La aplicación λ asigna a cada encargo de D la pila en la que se almacena y la condición (4.1) garantiza que el número de encargos asignados a cada pila no supera la capacidad máxima Q. Por ser π_1 y π_2 aplicaciones biyectivas de I en el conjunto de encargos D, éstas representan dos rutas que visitan todos los encargos, las cuales son compatibles entre sí y con la asignación de pilas por la condición (4.2), que hace que se respeten las relaciones de precedencia impuestas por la regla LIFO en cada pila. Por tanto, ambas condiciones son suficientes para asegurar la factibilidad de la solución.

$\Rightarrow$) Si (π_1, π_2, λ) es factible, no excede el tamaño máximo de las pilas, por lo que se verifica (4.1), y cumple las restricciones de precedencia, esto es cumple (4.2). □

En lo que sigue, cualquier solución factible S del DTSPMS se denotará por $S = (\pi_1, \pi_2, \lambda)$, para ciertas aplicaciones π_1, π_2, λ verificando las condiciones (4.1) y (4.2). Utilizando esta notación se definen todos los espacios de soluciones del problema.

Definición 19. El conjunto $X = \{(\pi_1, \pi_2, \lambda) \in Biy(I, D)^2 \times Apl(D, P) \mid (4.1), (4.2)\}$ se llama conjunto de soluciones **factibles** del DTSPMS.

Para la resolución del problema mediante heurísticas, en algunos casos resulta conveniente relajar la condición que limita el tamaño máximo de las pilas, permitiendo la utilización de cierto espacio extra con el objetivo de facilitar el proceso de búsqueda a través de soluciones intermedias no factibles. Si el margen permitido es de α unidades en cada pila, la condición que hay que imponer para garantizar que no se exceda la capacidad máxima viene dada por la expresión (4.3).

$$|\lambda^{-1}(k)| \leq Q + \alpha \qquad \forall k \in P \tag{4.3}$$

Las soluciones que cumplen esta condición relajada al introducir un margen α en las capacidades de las pilas forman un nuevo espacio de soluciones que se introduce a continuación.

Definición 20. Sea $\alpha \in \mathbb{N} \cup \{0\}$. El conjunto

$$X_\alpha = \{(\pi_1, \pi_2, \lambda) \in Biy(I, D)^2 \times Apl(D, P) \mid (4.3), (4.2)\}$$

se llama conjunto de soluciones α-**factibles** del DTSPMS. El número de unidades extra α añadidas a la capacidad máxima se denomina *margen*.

Una solución es α-factible si cumple las restricciones de precedencia y no supera la capacidad máxima de $Q + \alpha$ encargos por pila. La relación entre estas soluciones y las soluciones factibles definidas anteriormente se da en la siguiente proposición.

Proposición 5. *Se verifica que* $X \subset X_\alpha$, $\forall \alpha \in \mathbb{N} \cup \{0\}$, *es decir, toda solución factible es* α*-factible para cualquier* α. *Además, si* $Q < n$, *se tiene que* $X \supset X_\alpha$ *si y sólo si* $\alpha = 0$, *por lo que* $X = X_0$ *y los conceptos de solución factible y 0-factible son equivalentes.*

Además de relajar la capacidad máxima de las pilas, también se propondrán heurísticas que utilizarán soluciones intermedias que no verifican las restricciones de precedencia impuestas por el principio LIFO inherente a cada pila. Estas nuevas heurísticas requieren otro espacio de soluciones del problema, que se introduce a continuación.

Definición 21. Sea $\alpha \in \mathbb{N} \cup \{0\}$. El conjunto

$$\tilde{X}_\alpha = \{(\pi_1, \pi_2, \lambda) \in Biy(I, D)^2 \times Apl(D, P) \mid (4.3)\}$$

se llama conjunto de soluciones α-**potenciales** del DTSPMS. Una solución 0-potencial se denomina simplemente solución **potencial**.

Para que una solución sea α-potencial basta con que no exceda la capacidad máxima de $Q + \alpha$ encargos por pila. Forman el espacio de soluciones más grande, ya que contiene a los demás, según explicita la proposición 6.

Proposición 6. *Se verifica que* $X_\alpha \subset \tilde{X}_{\tilde{\alpha}}$, $\forall \alpha, \tilde{\alpha} \in \mathbb{N} \cup \{0\}$, $\alpha \leq \tilde{\alpha}$, *es decir, toda solución* α*-factible es* $\tilde{\alpha}$*-potencial si* $\alpha \leq \tilde{\alpha}$.

Función objetivo

Siguiendo la notación empleada en el Modelo 1, la distancia o coste de viaje entre las localizaciones de dos encargos $u, v \in D$ en la ruta $\delta \in \{1, 2\}$ se denota por c^δ_{uv} (el depósito central viene representado por la localización del encargo 0). Así, el coste de una solución $S = (\pi_1, \pi_2, \lambda)$, denotado por $z(S)$, es

$$z(S) = \sum_{\delta=1}^{2} \left(c^\delta_{0,\pi_\delta(1)} + \sum_{i=1}^{n-1} c^\delta_{\pi_\delta(i),\pi_\delta(i+1)} + c^\delta_{\pi_\delta(n),0} \right)$$

donde cada sumando en δ representa la distancia total recorrida o coste de viaje de la ruta δ: primero se va del depósito central (0) a la localización asociada al primer encargo $\pi_\delta(1)$,

después se visitan todas las demás según el orden dado por π_δ y finalmente se vuelve al depósito central desde la última localización visitada $\pi_\delta(n)$.

4.2. Generación de soluciones iniciales

Generar soluciones iniciales de forma adecuada es importante para el correcto funcionamiento de las heurísticas de mejora, ya que éstas precisan de una o varias soluciones de partida sobre las cuales iniciar la búsqueda de otras soluciones mejores. Generalmente estos algoritmos utilizan soluciones factibles durante todo el proceso de búsqueda y por tanto requieren soluciones iniciales factibles; sin embargo algunas heurísticas que se introducen en esta monografía se mueven en otros espacios de soluciones, no necesariamente factibles. Estos tipos de soluciones no necesariamente factibles deben cumplir un menor número de restricciones, lo cual permite diseñar métodos propios más rápidos y flexibles para su generación, aunque dichos métodos no sean válidos para generar soluciones factibles.

Se pretende elaborar métodos capaces de generar soluciones diversas utilizando el menor tiempo de cómputo posible, ya que se van a utilizar para la inicialización de otros algoritmos más sofisticados. No es necesario afinar demasiado en la calidad de las soluciones iniciales, pues en general una mejor solución inicial no siempre proporciona una mejor solución final; así, el objetivo principal será generar gran diversidad de soluciones con un gasto computacional pequeño, dejando como objetivo secundario la minimización de la función objetivo.

En las secciones siguientes se presentan distintos métodos para generar soluciones iniciales de todos los espacios de soluciones del problema siguiendo las pautas recién indicadas.

4.2.1. Soluciones iniciales factibles

La construcción de soluciones factibles para el DTSPMS no es inmediata, ya que es necesario garantizar el cumplimiento de ciertas restricciones que aseguren su factibilidad. Una solución factible se puede obtener de forma constructiva siguiendo una serie de pasos, determinísticos o aleatorios, adaptados al problema en cuestión, pero también se puede formar a partir de otra solución de un caso particular suyo más sencillo.

A continuación se presentan varios métodos para generar soluciones iniciales factibles del DTSPMS, que requieren poco tiempo de cómputo para su ejecución.

4.2.1.1. Resolución de un DTSPMS con una única pila

El caso particular del DTSPMS con $m = 1$ es relevante y por ello recibe una atención especial.

Definición 22. El *Doble Problema del Viajante con una Única Pila*, denotado por las siglas DTSPSS (del inglés *Double Traveling Salesman Problem with a Single Stack*), es un caso particular del DTSPMS en el que el número de pilas es $m = 1$. Dada una instancia (P) del DTSPMS se dice que la instancia (P') es el DTSPSS *asociado a* (P) si las distancias c_{ij}^{δ} son las de (P) y tiene una única pila con capacidad igual al número de encargos.

Si SS es una solución factible de un DTSPSS las rutas de entrega y recogida de SS son exactamente opuestas. Además, la asignación de pilas deja de formar parte de la solución del problema, ya que todos los encargos deben asignarse a la única pila disponible. Así, una solución del DTSPSS consta sólo de una ruta de entrega π_1 y una de recogida π_2, una opuesta de la otra, quedando determinada la solución por una cualquiera de ellas.

Sea (P) una instancia del DTSPMS con distancias c_{ij}^{δ} y (P') el DTSPSS asociado. (P') se reduce a un TSP, en el que para cada $i, j \in D$, la distancia entre el vértice i y el j es la suma de las distancias entre las localizaciones de los encargos i y j en los grafos de entrega y recogida, es decir,

$$d_{TSP}(i,j) = c_{ij}^{1} + c_{ij}^{2} \qquad \forall i, j \in D, i \neq j.$$

La solución de dicho TSP determina una de las rutas del DTSPSS, y la otra es la opuesta.

Una solución $SS = (\pi_1, \pi_2)$ del DTSPSS asociado (P') se puede transformar fácilmente en una solución $S = (\pi_1, \pi_2, \lambda)$ de (P) añadiendo cualquier asignación de pilas λ que no exceda la capacidad máxima de las mismas, ya que las relaciones de precedencia siempre se respetarán por ser π_2 exactamente opuesta a π_1.

Este proceso permite construir una solución factible del DTSPMS de cierta calidad resolviendo un TSP estándar, la cual puede utilizarse como solución inicial en cualquier algoritmo para el DTSPMS. En Petersen (2006) se aplica el algoritmo de los ahorros de Clarke y Wright (1964) al DTSPSS asociado para obtener una solución inicial del DTSPMS.

En este trabajo se aplica también el algoritmo de Clarke y Wright al DTSPSS asociado y la solución obtenida se mejora con un algoritmo de intercambio 2-óptimo (ver Croes, 1958).

4.2.1.2. Generación aleatoria dirigida

Una solución factible inicial para el DTSPMS puede generarse aleatoriamente de forma sencilla, dirigiendo la construcción de la solución para asegurar su factibilidad:

1. La ruta de recogida π_1 se puede generar de forma totalmente aleatoria, eligiendo al azar una permutación de los n encargos del problema.

2. Una vez construida la ruta π_1, la asignación de pilas λ se puede hacer de varias formas:

   a) Asignar cada encargo a una pila de forma aleatoria, descartando aquellas pilas que ya hayan alcanzado su capacidad máxima debido a las asignaciones realizadas con anterioridad.

   b) Siguiendo el orden marcado por la ruta π_1, asignar alternativamente cada encargo a una pila distinta. Así, el encargo recogido en el lugar i-ésimo se asigna a la pila $(i \bmod m) + 1$, es decir,

   $$\lambda(\pi_1(i)) = (i \bmod m) + 1, \quad \forall i \in I$$

   c) Siguiendo también el orden marcado por la ruta π_1, asignar los encargos a la misma pila hasta que ésta alcance su capacidad máxima. Cuando dicha pila esté llena, pasar a la siguiente y continuar el proceso. De esta forma, el encargo recogido en el lugar i-ésimo se asigna a la pila $\left\lfloor \frac{i}{Q} \right\rfloor + 1$, es decir,

   $$\lambda(\pi_1(i)) = \left\lfloor \frac{i}{Q} \right\rfloor + 1, \quad \forall i \in I$$

3. Construidas π_1 y λ, una ruta π_2 compatible con ellas también se puede generar de distintas formas:

   a) La ruta π_2 es exactamente opuesta a π_1. Al ser construida de esta forma, la ruta 2 siempre es compatible con la ruta 1 independientemente de la asignación de pilas λ.

   b) En la ruta π_2 los encargos asignados a cada pila se entregan de forma consecutiva: primero se entregan todos los encargos de la primera pila en el orden contrario al que fueron recogidos, después los de la siguiente, y así sucesivamente hasta que se terminan todas las pilas.

   c) La ruta π_2 se construye ordenadamente, eligiendo en cada momento cuál de los encargos situados en la cabecera de las pilas es el siguiente en ser entregado. En cada paso habrá a lo sumo m encargos disponibles entre los que elegir, pudiéndose realizar dicha elección de forma aleatoria u obedeciendo criterios tipo *greedy*: elegir el encargo más próximo, el más lejano, el de la pila más grande, etc.

4.2.2. Soluciones iniciales α-factibles

Los métodos presentados en la Sección 4.2.1 hacen referencia a la generación de soluciones factibles, en las que se asume que la capacidad máxima de las pilas en la instancia considerada es Q. Dichos métodos pueden extenderse de forma inmediata al espacio de soluciones α-factibles, para cualquier $\alpha \in \mathbb{N} \cup \{0\}$, simplemente considerando un tamaño máximo de $Q + \alpha$ unidades por pila, permitiendo la generación rápida de soluciones α-factibles.

4.2.3. Soluciones iniciales α-potenciales

Como toda solución factible ó α-factible es también α-potencial, los métodos anteriores también se pueden extender al espacio de soluciones α-potenciales. Sin embargo, como las soluciones α-potenciales no están obligadas a verificar las restricciones de precedencia, pueden diseñarse otros métodos específicos para generarlas que resulten más sencillos, rápidos y eficaces.

Al generar soluciones factibles ó α-factibles para el DTSPMS, el diseño de las rutas de la solución debe realizarse de forma coordinada para garantizar la existencia de una asignación de pilas compatible, lo cual produce inevitablemente un aumento en el coste final. Sin embargo, para generar soluciones α-potenciales esta coordinación no es necesaria, ya que no se requiere encontrar asignaciones de pilas factibles. Así, las rutas de recogida y entrega se pueden diseñar considerando de forma independiente los TSPs asociados a las regiones de entrega y recogida, respectivamente, y la asignación de pilas se diseña en función de ellas, obteniéndose soluciones con menor coste final a costa de introducir infactibilidad.

De esta manera, la idea es generar una pareja de rutas de *buena* calidad resolviendo independientemente con alguna heurística sencilla los dos TSP asociados a las dos regiones del problema, y posteriormente diseñar una asignación de pilas que cumpla el mayor número posible de restricciones de precedencia. Al igual que en la resolución del DTSPSS, en este trabajo se aplica el algoritmo de Clarke y Wright a cada uno de los dos TSP independientes y las soluciones obtenidas son mejoradas con un algoritmo de intercambio 2-óptimo (Croes, 1958).

Dada una pareja de rutas de recogida y entrega, la asignación de pilas se genera con el algoritmo de Asignación de Pilas Dirigida (DSA: *Directed Stack Assignment*), que se presenta a continuación. Este algoritmo asigna una pila a cada encargo con la intención de que la solución α-potencial final obtenida esté próxima a la factibilidad, utilizando poco tiempo de cómputo.

Algoritmo 12. Asignación de Pilas Dirigida (DSA)

Parámetros de entrada

- π_1, π_2: Rutas de recogida y entrega de la solución.

Parámetros de salida

- λ: Asignación de pilas diseñada.

Pseudocódigo

1. *Inicialización:* Poner $i = 1$, $P^* = P$.
2. *Elección de pila:* Elegir la pila $k \in P^*$ tal que al definir $\lambda(\pi_1(i)) = k$ la infactibilidad de la solución (π_1, π_2, λ) parcial aumente lo menos posible. En caso de empate entre varias pilas, elegir aquella con menor número de encargos asignados a ella.
3. *Asignación de pila:* Definir $\lambda(\pi_1(i)) = k$.
4. *Actualización:* Poner $i = i + 1$; si $|\lambda^{-1}(k)| = Q + \alpha$ poner $P^* = P^* \setminus \{k\}$.
5. *Criterio de parada:*
   - Si $i \leq n$ volver al paso 2.
   - Si $i > n$ FIN: la asignación de pilas diseñada viene dada por la aplicación $\lambda : D \longrightarrow P$.

4.3. Estructuras de entornos para soluciones factibles

La mayoría de los algoritmos heurísticos utilizados para resolver problemas de optimización combinatoria incluyen de alguna manera un proceso de búsqueda local, el cual siempre lleva implícita la definición de una estructura de entornos adaptada al problema a resolver que permita pasar de una solución a otra solución vecina de mayor calidad. La flexibilidad del proceso de búsqueda, su capacidad para encontrar buenas soluciones, la complejidad de los movimientos a realizar y en definitiva el funcionamiento de todo el proceso depende fuertemente de la estructura de entornos utilizada. Cada estructura de entornos define un conjunto de óptimos locales diferente, al cual puede pertenecer el óptimo global o no. Por tanto, el diseño de estructuras de entornos flexibles y variadas que permitan explorar grandes zonas del espacio factible y proporcionen soluciones que aúnen calidad y diversidad es crucial a la hora de diseñar un procedimiento de búsqueda local para cualquier heurística.

En las siguientes secciones se presentan siete estructuras de entornos distintas para el DTSPMS, las cuales permiten la realización de búsquedas locales bajo diversos criterios dentro del espacio de soluciones factibles del problema. Las dos primeras, denominadas *Intercambio en Ruta* e *Intercambio Entre Pilas*, fueron introducidas en Petersen (2006), donde fueron utilizadas para adaptar heurísticas como la Búsqueda Tabú y el Temple Simulado a la resolución del DTSPMS; las cinco restantes, denominadas *Intercambio Dentro de una Pila*, *Reinserción*, *r-Permutación en Ruta*, *r-Permutación en Pila* y *r-Permutación Completa en Pila*, se introducen aquí por primera vez.

La utilización conjunta de las estructuras de entornos que se presentan en las Secciones 4.3.1-4.3.7 permite explorar las zonas más prometedoras del espacio de soluciones factibles realizando movimientos que modifican tanto las rutas como la asignación de pilas de las soluciones utilizadas. Todas estas estructuras y los operadores que las determinan se definen de manera que se garantice la factibilidad de las soluciones proporcionadas; sin embargo, en la Sección 4.3.8 se indica cómo todas ellas se pueden extender fácilmente para considerar también soluciones α-factibles.

4.3.1. Intercambio en Ruta

El entorno *Intercambio en Ruta* (*Route Swap*, RS) de una solución S del DTSPMS, denotado por RS(S), está formado por todas aquellas soluciones que se pueden obtener a partir de S intercambiando las posiciones de dos clientes consecutivos en alguna de las dos rutas. Un movimiento a través del cual se genera una solución del entorno RS(S) a partir de S se denomina por extensión Movimiento de Intercambio en Ruta y se denota por RS.

Al realizar un movimiento RS sólo se modifica la parte de la solución relativa a las rutas recorridas, manteniéndose intacta la asignación de pilas. Un movimiento de este tipo puede realizarse tanto en la ruta de entrega como en la ruta de recogida, actualizando convenientemente la otra ruta, si es necesario, para mantener la factibilidad de la solución. La necesidad de modificar la ruta sobre la cual no se realiza el movimiento depende de si los encargos asociados a los vértices a intercambiar están almacenados en la misma pila o en pilas distintas, como se estudia a continuación.

Factibilidad de la solución

Permutar las posiciones de dos encargos consecutivos en una de las rutas de una solución del DTSPMS puede dar lugar a soluciones no factibles que violan las relaciones de precedencia. Por ello, en un movimiento RS resulta necesario realizar ciertas operaciones para mantener la factibilidad de la solución después del intercambio de dos encargos. Dichas operaciones vienen dadas por la proposición 7.

Proposición 7. *Sean $u, v \in D$ los dos encargos visitados de forma consecutiva en la ruta de recogida que se van a intercambiar en un movimiento* RS *sobre la Ruta de Recogida (ruta 1), y sea $S = (\pi_1, \pi_2, \lambda)$ la solución inicial sobre la que se realiza el movimiento. La solución S' obtenida a partir de S intercambiando las posiciones de u y v en la ruta 1 verifica (i) o (ii):*

(i.) Si los encargos u y v se almacenan en la misma pila, S' puede no ser factible. Si en la solución S' se intercambian las posiciones de los encargos u y v también en la ruta 2 entonces S' siempre es factible.

(ii.) Si los encargos u y v se almacenan en pilas distintas, S' es una solución factible del DTSPMS.

Demostración

Sea $u \in D$ el encargo que se recoge inmediatamente antes que $v \in D$. Hay dos casos posibles:

i. Los encargos u y v se almacenan en la misma pila. En este caso es $\lambda(u) = \lambda(v) = k$. Como u y v son consecutivos en la ruta 1, u se almacena inmediatamente antes que v en la pila k, debiéndose entregar v antes que u. Al intercambiar sus posiciones en la ruta 1, su orden relativo en la pila k se modifica, de manera que, después del movimiento, primero se almacena el encargo v e inmediatamente después el u. Como consecuencia de este cambio en el orden de almacenamiento de la pila k, el encargo u pasa a estar disponible para su entrega antes que el v, y por ello las posiciones de los clientes u y v en la Ruta de Entrega (ruta 2) también deben ser intercambiadas (aunque dichos clientes no sean consecutivos en la ruta 2), para mantener la factibilidad de la solución.

ii. Los encargos u y v se almacenan en pilas distintas. Sean $p = \lambda(u)$, $q = \lambda(v)$ las pilas asignadas a los encargos u y v, respectivamente. Como v se visita inmediatamente después que u en la ruta 1 y ambos están asignados a distintas pilas, en el momento en que se visita u y se almacena en la pila p, la posición de carga del encargo v en la pila q también está disponible. Así, al realizar el movimiento e intercambiar las posiciones de u y v en la ruta 1, el orden relativo de los encargos en las pilas p y q sigue siendo el mismo, por lo que la antigua ruta 2 sigue siendo factible y no es necesario modificarla.

Con lo que queda demostrada la proposición. □

Observación 6. El movimiento RS se puede realizar sobre la Ruta de Entrega en lugar de sobre la Ruta de Recogida de manera análoga.

Observación 7. Conviene resaltar que, aunque el orden relativo de los encargos en cada pila puede variar como consecuencia de los intercambios de posiciones en las rutas, la asignación de pilas (dada por la aplicación λ), que indica en qué pila se almacena cada encargo, no se ve afectada por un movimiento de este tipo.

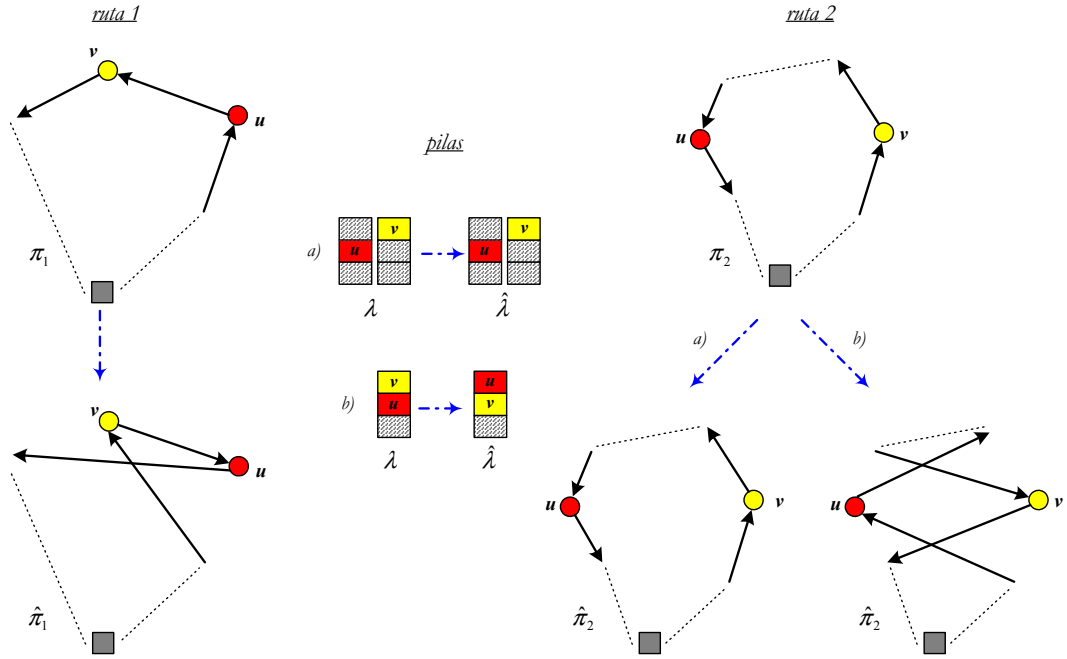


Figura 4.1: Un movimiento de Intercambio en Ruta

En la Figura 4.1 se ilustra cómo se realiza un movimiento RS sobre la ruta 1 de la solución (π_1, π_2, λ) en la que se intercambian los encargos consecutivos u y v. La ruta $\hat{\pi}_1$ se construye en cualquier caso intercambiando los encargos u y v, mientras que para la ruta $\hat{\pi}_2$ se deben distinguir los dos casos tratados en la proposición 7: en el caso *a)* los encargos u y v están en distintas pilas y por tanto la ruta π_2 y la asignación de pilas λ no cambian, mientras que en el caso *b)* dichos encargos están en la misma pila y por ello en la ruta $\hat{\pi}_2$ y en la asignación de pilas $\hat{\lambda}$ es necesario intercambiarlos.

Realización de un movimiento

Un movimiento de Intercambio en Ruta queda determinado por la ruta sobre la cual realizar el intercambio y el primer encargo a intercambiar. La definición 23 precisa este tipo de movimientos.

Definición 23. Un *movimiento de Intercambio en Ruta* sobre una solución S del DTSPMS, denotado por RS, consiste en intercambiar las posiciones de dos encargos visitados consecutivamente en una ruta. Sean $\delta \in \{1,2\}$ la ruta y $u \in D$ el encargo elegidos. Se define el conjunto $X^{RS}_{u,\delta}$ formado por las soluciones sobre las que se puede realizar el movimiento RS que queda determinado por u y δ de la siguiente manera:

$$X^{RS}_{u,\delta} = \left\{(\pi_1, \pi_2, \lambda) \in X \,|\, \pi_\delta^{-1}(u) \neq n\right\}.$$

Sobre este conjunto se define el *operador movimiento*

$$\begin{array}{rccl} \mathrm{RS}[u,\delta] : X^{RS}_{u,\delta} & \longrightarrow & X \\ S & \longmapsto & \mathrm{RS}[u,\delta](S) \equiv \text{solución tras movimiento RS} \end{array}$$

que asigna a cada solución S de su dominio la solución que se obtiene realizando sobre S el movimiento de Intercambio en Ruta sobre la ruta δ en la que se intercambian los encargos u y v, siendo $v = \pi_\delta(\pi_\delta^{-1}(u)+1)$ el encargo visitado inmediatamente después que u en la ruta δ.

Proposición 8. *Sean $\delta \in \{1,2\}$, $\gamma \in \{1,2\}\backslash\{\delta\}$ y $S = (\pi_1, \pi_2, \lambda) \in X^{RS}_{u,\delta}$. La solución $\hat{S} = (\hat{\pi}_1, \hat{\pi}_2, \hat{\lambda}) = \mathrm{RS}[u,\delta](S)$ es la siguiente:*

$$\hat{\pi}_\delta(i) = \begin{cases} \pi_\delta(i) & si \quad i \in I\backslash\{i_0, i_0+1\} \\ \pi_\delta(i_0+1) & si \quad i = i_0 \\ \pi_\delta(i_0) & si \quad i = i_0+1 \end{cases} \qquad \hat{\lambda}(w) = \lambda(w), \quad \forall w \in D$$

- *Si $\lambda(u) \neq \lambda(v)$:* $\quad \hat{\pi}_\gamma(i) = \pi_\gamma(i), \quad \forall i \in I$
- *Si $\lambda(u) = \lambda(v)$:*

$$\hat{\pi}_\gamma(i) = \begin{cases} \pi_\gamma(i) & si \quad i \in I\backslash\{i_1, i_2\} \\ \pi_\gamma(i_2) & si \quad i = i_1 \\ \pi_\gamma(i_1) & si \quad i = i_2 \end{cases}$$

donde $i_0 = \pi_\delta^{-1}(u)$, $v = \pi_\delta(i_0+1)$, $i_1 = \pi_\gamma^{-1}(u)$, $i_2 = \pi_\gamma^{-1}(v)$.

Observación 8. La definición de las aplicaciones $\hat{\pi}_\delta$ y $\hat{\lambda}$ siempre es la misma, mientras que $\hat{\pi}_\gamma$ se define de una u otra forma según los encargos a intercambiar estén o no en la misma pila.

Representación conjuntista del entorno

El entorno RS de una solución del DTSPMS es el conjunto formado por todas las soluciones que pueden obtenerse a partir de ella realizando un movimiento de Intercambio en Ruta. En la proposición 9 se detalla cómo construir este tipo de entornos.

Proposición 9. *Sean* $S=(\pi_1,\pi_2,\lambda)\in X$ *una solución factible del DTSPMS y*

$$X^{\lambda}=\{(\hat{\pi}_1,\hat{\pi}_2,\hat{\lambda})\in X|\hat{\lambda}=\lambda\}\subset X \tag{4.4}$$

el conjunto de soluciones factibles con la misma asignación de pilas que S*. El entorno Intercambio en Ruta de la solución* S*, denotado por* $\mathrm{RS}(S)$*, es el conjunto*

$$\mathrm{RS}(S)=RS_1^a\cup RS_1^b\cup RS_2^a\cup RS_2^b$$

donde

$RS_\delta^a=\left\{(\hat{\pi}_1,\hat{\pi}_2,\hat{\lambda})\in X^{\lambda}\,\middle|\,\gamma\in\{1,2\}\backslash\{\delta\},\ \hat{\pi}_\gamma=\pi_\gamma\ y\ \exists i_0\in I\backslash\{n\}\ \text{sujeto a}\ C_1^a,C_2^a,C_3^a\right\}$ *para* $\delta\in\{1,2\}$ *y las condiciones* C_i^a*,* $i=1,2,3$*, son:*

i. $C_1^a:\lambda\big(\hat{\pi}_\delta(i_0)\big)\neq\lambda\big(\hat{\pi}_\delta(i_0+1)\big)$

ii. $C_2^a:\hat{\pi}_\delta(i_0)=\pi_\delta(i_0+1),\ \hat{\pi}_\delta(i_0+1)=\pi_\delta(i_0)$

iii. $C_3^a:\hat{\pi}_\delta(i)=\pi_\delta(i)\ \forall i\in I\backslash\{i_0,i_0+1\}$

$RS_\delta^b=\left\{(\hat{\pi}_1,\hat{\pi}_2,\hat{\lambda})\in X^{\lambda}\,\middle|\,\gamma\in\{1,2\}\backslash\{\delta\},\ \exists i_0\in I\backslash\{n\}\ \text{sujeto a}\ C_1^b,C_2^b,C_3^b\right\}$ *para* $\delta\in\{1,2\}$ *y las condiciones* C_i^b*,* $i=1,2,3$*, son:*

i. $C_1^b:\lambda\big(\hat{\pi}_\delta(i_0)\big)=\lambda\big(\hat{\pi}_\delta(i_0+1)\big)$

ii. $C_2^b:\hat{\pi}_\delta(i_0)=\pi_\delta(i_0+1),\ \hat{\pi}_\delta(i_0+1)=\pi_\delta(i_0),\ \hat{\pi}_\delta(i)=\pi_\delta(i)\ \forall i\in I\backslash\{i_0,i_0+1\}$

iii. $C_3^b:\hat{\pi}_\gamma(i_1)=\pi_\gamma(i_2),\ \hat{\pi}_\gamma(i_2)=\pi_\gamma(i_1),\ \hat{\pi}_\gamma(i)=\pi_\gamma(i)\ \forall i\in I\backslash\{i_1,i_2\}$
$i_1=\pi_\gamma^{-1}(\pi_\delta(i_0)),\ i_1=\pi_\gamma^{-1}(\pi_\delta(i_0+1))$.

Demostración

Los conjuntos RS_1^a e RS_1^b, con subíndice 1, están formados por las soluciones obtenidas con un movimiento RS sobre la ruta 1 de S, mientras que los conjuntos RS_2^a e RS_2^b, con subíndice 2, contienen soluciones obtenidas a partir de un movimiento RS sobre la ruta 2 de S.

Los conjuntos RS_1^a e RS_2^a, con superíndice a, están formados por las soluciones obtenidas con un movimiento RS sobre S en el cual los encargos intercambiados están asignados

a pilas distintas, de manera que la ruta sobre la que no se realiza el movimiento permanece intacta. Los conjuntos RS_1^b e RS_2^b, con superíndice b, contienen soluciones obtenidas a partir de un movimiento RS sobre S en el que los encargos intercambiados están asignados a la misma pila, de forma que las dos rutas de la solución se ven modificadas.

La condición C_1^a indica que los encargos intercambiados están asignados a pilas distintas; la condición C_2^a asegura que, en efecto, las posiciones de los encargos elegidos están intercambiadas en la ruta sobre la que se realiza el movimiento; la condición C_3^a implica que las posiciones de los encargos no intercambiados se mantienen intactas en la ruta sobre la que se realiza el movimiento.

La condición C_1^b indica que los encargos intercambiados están asignados a la misma pila; la condición C_2^b implica que las posiciones de los encargos elegidos están intercambiadas en la ruta sobre la que se realiza el movimiento y las del resto se mantienen intactas; la condición C_3^b asegura que también se intercambian las posiciones de los encargos elegidos y las demás no se modifican en la ruta sobre la que no se realiza el movimiento, la cual es necesario actualizar por estar los encargos intercambiados asignados a la misma pila. □

Tamaño del entorno

El tamaño de un entorno de Intercambio en Ruta es lineal con respecto al número n de encargos del problema, como se detalla a continuación.

Proposición 10. *Sea $S \in X$ una solución factible del DTSPMS. El entorno* $\mathrm{RS}(S)$ *tiene tamaño $O(n)$.*

Demostración

Para cada $\delta \in \{1, 2\}$, un movimiento RS sobre la ruta δ de la solución $S = (\pi_1, \pi_2, \lambda)$ viene determinado por el primero de los encargos $u \in D$ que se van a intercambiar, ya que el segundo encargo $v \in D$ es $v = \pi_\delta(\pi_\delta^{-1}(u) + 1)$.

El conjunto $U = \{u \in D | \pi_\delta^{-1}(u) \neq n\}$ es el conjunto de todos los encargos que determinan un movimiento RS sobre la ruta δ, puesto que se puede elegir cualquiera menos el que ocupa la última posición de la ruta. Así, hay $|U| = n - 1$ movimientos RS distintos sobre cada ruta.

El número total de movimientos RS posibles sobre una solución inicial es, por tanto, $2(n-1)$, esto es $O(n)$. □

Observación 9. Según la notación introducida en la proposición 9, los conjuntos $(RS_1^a \cup RS_1^b)$ y $(RS_2^a \cup RS_2^b)$ representan las distintas soluciones que se pueden obtener realizando

movimientos del tipo RS sobre las rutas 1 y 2, respectivamente. Así, se tiene que $|RS_1^a \cup RS_1^b| = |RS_2^a \cup RS_2^b| = n - 1$. Sin embargo, los cardinales de los conjuntos RS_δ^a y RS_δ^b, $\delta \in \{1, 2\}$, no pueden calcularse por separado, ya que dependen del número de clientes consecutivos en las rutas que estén asignados a pilas iguales.

4.3.2. Intercambio Entre Pilas

El entorno de *Intercambio Entre Pilas* (*Complete Swap*, CS) de una solución S del DTSPMS, denotado por CS(S), está formado por todas aquellas soluciones que se pueden obtener a partir de S intercambiando las posiciones en las pilas de dos encargos asignados a pilas distintas. La motivación de un movimiento de este tipo, denominado por extensión movimiento de Intercambio Entre Pilas y denotado por CS, es modificar exclusivamente la asignación de pilas de S, pero para mantener la factibilidad de la solución es necesario realizar también algunas modificaciones en las rutas.

Factibilidad de la solución

Intercambiar las pilas asignadas a dos encargos en una solución del DTSPMS puede dar lugar a soluciones no factibles por no respetar las relaciones de precedencia. Por ello, en un movimiento CS resulta necesario realizar ciertas operaciones adicionales para mantener la factibilidad de la solución. Dichas operaciones vienen dadas por la proposición 11:

Proposición 11. *Sean $u, v \in D$ los dos encargos asignados a distintas pilas que se van a intercambiar en un movimiento* CS. *Sea $S = (\pi_1, \pi_2, \lambda)$ la solución inicial sobre la que se realiza el movimiento y S' la solución obtenida a partir de S intercambiando las pilas asignadas a los encargos u y v. Si se intercambian las posiciones de los encargos u y v en las rutas 1 y 2 entonces S' es fatible.*

Demostración

Al realizarse el movimiento CS se intercambian las pilas a las que están asignados los encargos u y v y las rutas π_1 y π_2 dejarán de ser, en general, compatibles con la nueva asignación de pilas, ya que al cambiar de pila es posible que se dejen de respetar las condiciones de precedencia con respecto a las nuevas pilas. Por ello, si se intercambian las posiciones de los encargos u y v en las rutas de entrega y recogida se asegura la compatibilidad entre las rutas y la nueva asignación de pilas. □

En la Figura 4.2 se ilustra cómo realizar un movimiento CS sobre una solución (π_1, π_2, λ) en el que se intercambian dos encargos u y v asignados a distintas pilas. En la representación

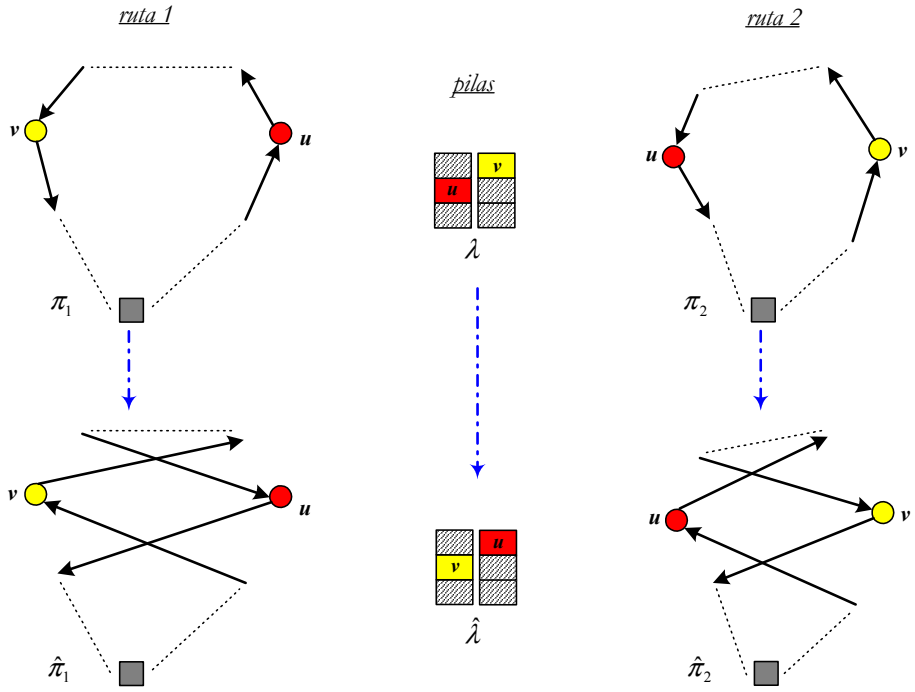


Figura 4.2: Un movimiento de Intercambio Entre Pilas

de π_2 se asume que el encargo v se entrega antes que u, pero también podría darse la situación contraria, en la que el movimiento se realizaría de manera análoga. La nueva solución obtenida viene representada por $(\hat{\pi}_1, \hat{\pi}_2, \hat{\lambda})$.

Realización de un movimiento

Un movimiento de Intercambio entre Pilas queda determinado por los dos encargos, asignados a distintas pilas, que se van a intercambiar. La definición 24 precisa este tipo de movimientos.

Definición 24. Un *movimiento de Intercambio Entre Pilas* sobre una solución del DTSPMS, denotado por CS, consiste en intercambiar las posiciones de dos encargos asignados a distintas pilas. Sean $u, v \in D$ dos encargos del problema. El conjunto $X^{CS}_{u,v}$ formado por las soluciones sobre las que se puede realizar el movimiento CS que queda determinado por u y v se define como:

$$X^{CS}_{u,v} = \{(\pi_1, \pi_2, \lambda) \in X \mid \lambda(u) \neq \lambda(v)\}\,.$$

Sobre este conjunto se define el *operador movimiento*

$$\begin{array}{rcl} \mathrm{CS}[u,v] : X_{u,v}^{CS} & \longrightarrow & X \\ S & \longmapsto & \mathrm{CS}[u,v](S) \equiv \text{solución tras movimiento CS} \end{array}$$

que asigna a cada solución S de su dominio la solución que se obtiene realizando sobre S el movimiento de Intercambio Entre Pilas en el que se intercambian los encargos u y v.

Definición 25. Sean A, B conjuntos finitos, $f : A_f \longrightarrow B_f$, A_f el dominio de f ($A_f \subset A$) y B_f el recorrido de f ($B_f \subset B$). Dados $i, j \in A$ se define F_i^j como

$$F_i^j = \{f : A_f \longrightarrow B_f \mid i, j \in A_f\}$$

esto es, el conjunto formado por todas las aplicaciones cuyo dominio contiene a los elementos i, j. Se define el operador

$$\begin{array}{rcl} \mathcal{I}_i^j : F_i^j & \longrightarrow & F_i^j \\ f & \longmapsto & \mathcal{I}_i^j(f) : A_f \longrightarrow B_f \end{array}$$

que asigna a cada aplicación $f \in F_i^j$ otra aplicación $g = \mathcal{I}_i^j(f)$ con el mismo dominio y recorrido que f y definida como

$$g(i) = f(j),\ g(j) = f(i) \ \text{ y } \ g(k) = f(k)\ \forall k \in A_f \backslash \{i, j\}$$

es decir, las aplicación g coincide con f en todos los elementos del dominio excepto i y j, y las imágenes de estos dos elementos están intercambiadas.

Observación 10. El operador $\mathcal{I}_i^j$ introducido en la definición 25 facilita la obtención de rutas construidas intercambiando dos posiciones: la ruta $\hat{\pi}$ construida a partir de π intercambiando los encargos que ocupan las posiciones i y j se define simplemente como $\hat{\pi} = \mathcal{I}_i^j(\pi)$.

Observación 11. Los operadores $\mathcal{I}_i^j(\pi)$ y $\mathcal{I}_j^i(\pi)$ coinciden para cualesquiera $i, j \in A$.

La forma de construir la solución obtenida tras un movimiento CS viene dada por la proposición 12.

Proposición 12. *La solución* $\hat{S} = (\hat{\pi}_1, \hat{\pi}_2, \hat{\lambda}) = \mathrm{CS}[u,v](S)$ *resultado de aplicar el operador movimiento* $\mathrm{CS}[u,v]$ *sobre una solución* $S = (\pi_1, \pi_2, \lambda) \in X_{u,v}^{CS}$ *es la siguiente:*

$$\hat{\pi}_1(i) = \begin{cases} \pi_1(i) & si \quad i \in I\backslash\{i_1, j_1\} \\ \pi_1(j_1) & si \quad i = i_1 \\ \pi_1(i_1) & si \quad i = j_1 \end{cases} \qquad \hat{\pi}_2(i) = \begin{cases} \pi_2(i) & si \quad i \in I\backslash\{i_2, j_2\} \\ \pi_2(j_2) & si \quad i = i_2 \\ \pi_2(i_2) & si \quad i = j_2 \end{cases}$$

$$\hat{\lambda}(w) = \begin{cases} \lambda(w) & si \quad w \in D \backslash \{u, v\} \\ \lambda(v) & si \quad w = u \\ \lambda(u) & si \quad w = v \end{cases}$$

donde $i_1 = \pi_1^{-1}(u)$, $i_2 = \pi_2^{-1}(u)$ *son las posiciones del encargo* u *en las rutas 1 y 2, respectivamente, y* $j_1 = \pi_1^{-1}(v)$, $j_2 = \pi_2^{-1}(v)$ *las del encargo* v.

Observación 12. De forma equivalente, utilizando el operador introducido en la definición 25, los elementos de la solución $\hat{S} = (\hat{\pi}_1, \hat{\pi}_2, \hat{\lambda})$ se pueden definir abreviadamente como

$$\hat{\pi}_1 = \mathcal{I}_{i_1}^{j_1}(\pi_1), \;\; \hat{\pi}_2 = \mathcal{I}_{i_2}^{j_2}(\pi_2), \;\; \hat{\lambda} = \mathcal{I}_u^v(\lambda).$$

Observación 13. En un movimiento CS la forma en la que se definen las nuevas aplicaciones $\hat{\pi}_1$, $\hat{\pi}_2$, y $\hat{\lambda}$ es independiente de la solución inicial S considerada.

Representación conjuntista del entorno

El entorno CS de una solución del DTSPMS es el conjunto formado por todas las soluciones que pueden obtenerse a partir de ella realizando un movimiento de Intercambio Entre Pilas. En la proposición 13 se detalla cómo construir este tipo de entornos.

Proposición 13. *Sea* $S = (\pi_1, \pi_2, \lambda) \in X$ *una solución factible del DTSPMS. El entorno Intercambio Entre Pilas de la solución* S*, denotado por* $\mathrm{CS}(S)$*, es el conjunto*

$$\mathrm{CS}(S) = \left\{ (\hat{\pi}_1, \hat{\pi}_2, \hat{\lambda}) \in X \;\middle|\; \exists u, v \in D \text{ s.a. } \lambda(u) \neq \lambda(v),\, \hat{\lambda} = \mathcal{I}_u^v(\lambda),\, \hat{\pi}_1 = \mathcal{I}_{i_1}^{j_1}(\pi_1),\, \hat{\pi}_2 = \mathcal{I}_{i_2}^{j_2}(\pi_2) \right\}$$

donde $i_1 = \pi_1^{-1}(u)$, $i_2 = \pi_2^{-1}(u)$, $j_1 = \pi_1^{-1}(v)$, $j_2 = \pi_2^{-1}(v)$.

Tamaño del entorno

El tamaño de un entorno de Intercambio Entre Pilas es cuadrático con respecto al número n de encargos del problema, según se detalla a continuación.

Proposición 14. *Sea* $S \in X$ *una solución factible del DTSPMS. El entorno* $\mathrm{CS}(S)$ *es a lo sumo de tamaño* $O(n^2)$.

Demostración

Un movimiento CS sobre una solución $S = (\pi_1, \pi_2, \lambda)$ viene determinado por la pareja de encargos $u, v \in D$ asignados a distintas pilas cuyas posiciones se intercambian. El número

de parejas de este tipo depende de las características de la solución S, más concretamente, del número de encargos asignados a cada pila.

Para cada $k \in P$ sea $n_k = |\lambda^{-1}(k)|$ el tamaño de la pila k en la solución S, es decir, el número de encargos asignados a la pila k. En total hay $\binom{n}{2}$ parejas de encargos, de las cuales $\binom{n_k}{2}$ están asignadas a la misma pila $k \in P$. Por tanto, hay $\binom{n}{2} - \sum_{k=1}^{m} \binom{n_k}{2}$ parejas de encargos asignados a pilas distintas.

Así, el entorno CS(S) es a lo sumo de tamaño

$$O\left(\binom{n}{2} - \sum_{k=1}^{m} \binom{n_k}{2}\right) = O\left(\binom{n}{2}\right) = O(n^2).$$

□

4.3.3. Intercambio Dentro de una Pila

El entorno de *Intercambio Dentro de una Pila* (*In-Stack Swap*, ISS) de una solución S del DTSPMS, denotado por ISS(S), está formado por todas aquellas soluciones que se pueden obtener a partir de S intercambiando las posiciones de dos encargos asignados a la misma pila. La motivación de un movimiento de este tipo, denominado por extensión movimiento de Intercambio Dentro de una Pila y denotado por ISS, es modificar el orden relativo de los encargos elegidos en la pila a la que están asignados, sin cambiarlos de pila y por tanto sin modificar la asignación de pilas de S. Para conseguir este efecto y mantener la factibilidad de la solución, lo que hay que hacer es intercambiar las posiciones de los encargos en las dos rutas de S, para que así el orden en que se recoge y se entrega la mercancía asignada a dichos encargos se intercambie y por extensión también lo haga el orden de almacenamiento en la pila k.

Un movimiento ISS es similar a uno del tipo CS, ya que realiza operaciones parecidas sobre la solución de partida, intercambiando las posiciones de los encargos elegidos en las dos rutas de la solución. La diferencia principal estriba en el hecho de que en un movimiento CS los encargos elegidos están asignados a pilas distintas, y por tanto al intercambiarlos también se modifica la asignación de pilas; sin embargo, en un movimiento ISS, el intercambio se realiza entre encargos asignados a la misma pila, lo cual provoca un cambio en el orden relativo en el que se almacenan los encargos dentro de la pila, pero sin alterar la asignación de pilas.

Factibilidad de la solución

Proposición 15. *Sea $S = (\pi_1, \pi_2, \lambda)$ una solución del DTSPMS y sean $u, v \in D$ dos encar-*

gos asignados a la misma pila en S. La solución S' obtenida a partir de S intercambiando las posiciones de los encargos u y v en las dos rutas de la solución para intercambiar sus posiciones de almacenamiento en la pila k y manteniendo la asignación de pilas siempre es factible.

Observación 14. Si i_1, i_2 denotan las posiciones del encargo u en las rutas 1 y 2, respectivamente, y j_1, j_2 las del encargo v, el movimiento consiste en intercambiar las imágenes de i_1 y j_1 por la aplicación π_1 y las imágenes de i_2 y j_2 por π_2.

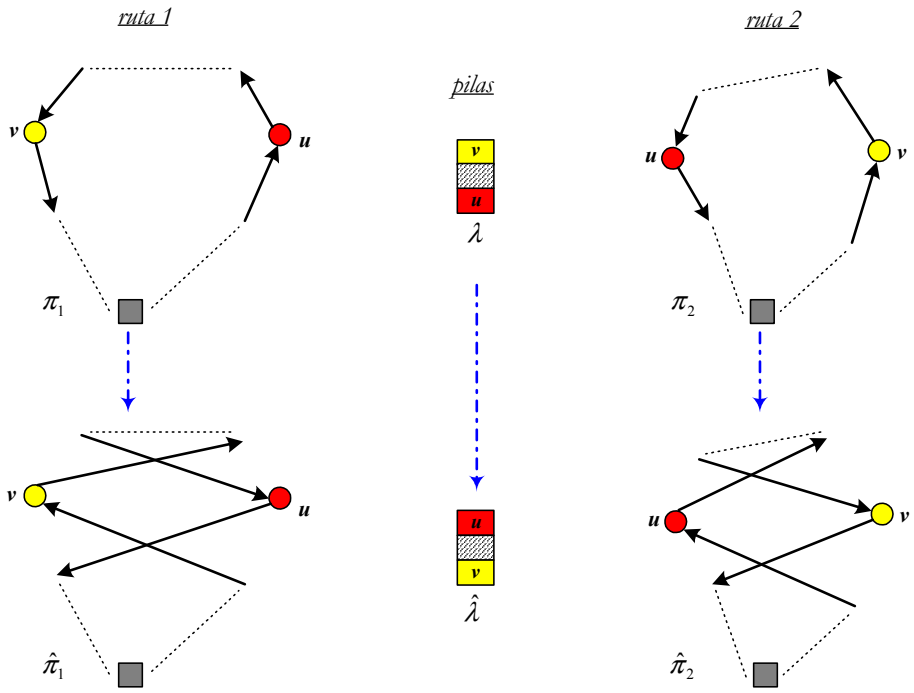


Figura 4.3: Un movimiento de Intercambio Dentro de una Pila

La Figura 4.3 ilustra cómo realizar un movimiento ISS sobre una solución (π_1, π_2, λ) en el que se intercambian dos encargos u y v asignados a la misma pila. La nueva solución obtenida viene representada por $(\hat{\pi}_1, \hat{\pi}_2, \hat{\lambda})$.

Realización de un movimiento

Un movimiento ISS queda determinado por los dos encargos, asignados a la misma pila, que se van a intercambiar. La definición 26 precisa este tipo de movimientos.

Definición 26. Un *movimiento de Intercambio Dentro de una Pila* sobre una solución del DTSPMS, denotado por ISS, consiste en intercambiar las posiciones de dos encargos

asignados a la misma pila. Sean $u, v \in D$ dos encargos del problema. El conjunto $X_{u,v}^{ISS}$ formado por las soluciones sobre las que se puede realizar el movimiento ISS que queda determinado por u y v se define como:

$$X_{u,v}^{ISS} = \{(\pi_1, \pi_2, \lambda) \in X \,|\, \lambda(u) = \lambda(v)\}\,.$$

Sobre este conjunto se define el *operador movimiento*

$$\begin{array}{rcl} \mathrm{ISS}[u,v] : X_{u,v}^{ISS} & \longrightarrow & X \\ S & \longmapsto & \mathrm{ISS}[u,v](S) \equiv \text{solución tras movimiento ISS} \end{array}$$

que asigna a cada solución S de su dominio la solución que se obtiene realizando sobre S el movimiento de Intercambio Dentro de una Pila en el que se intercambian los encargos u y v.

La forma de construir la solución obtenida tras un movimiento ISS viene dada por la proposición 16.

Proposición 16. *La solución* $\hat{S} = (\hat{\pi}_1, \hat{\pi}_2, \hat{\lambda}) = \mathrm{ISS}[u,v](S)$ *resultado de aplicar el operador movimiento* $\mathrm{ISS}[u,v]$ *sobre una solución* $S = (\pi_1, \pi_2, \lambda) \in X_{u,v}^{ISS}$ *es la siguiente:*

$$\hat{\pi}_1(i) = \begin{cases} \pi_1(i) & si \quad i \in I \backslash \{i_1, j_1\} \\ \pi_1(j_1) & si \quad i = i_1 \\ \pi_1(i_1) & si \quad i = j_1 \end{cases} \qquad \hat{\pi}_2(i) = \begin{cases} \pi_2(i) & si \quad i \in I \backslash \{i_2, j_2\} \\ \pi_2(j_2) & si \quad i = i_2 \\ \pi_2(i_2) & si \quad i = j_2 \end{cases}$$

$$\hat{\lambda}(w) = \lambda(w), \quad \forall w \in D$$

donde $i_1 = \pi_1^{-1}(u)$, $i_2 = \pi_2^{-1}(u)$ *son las posiciones del encargo* u *en las rutas 1 y 2, respectivamente, y* $j_1 = \pi_1^{-1}(v)$, $j_2 = \pi_2^{-1}(v)$ *las del encargo* v.

Observación 15. De forma equivalente, utilizando el operador introducido en la definición 25, los elementos de la solución $\hat{S} = (\hat{\pi}_1, \hat{\pi}_2, \hat{\lambda})$ se pueden definir abreviadamente como

$$\hat{\pi}_1 = \mathcal{I}_{i_1}^{j_1}(\pi_1), \;\; \hat{\pi}_2 = \mathcal{I}_{i_2}^{j_2}(\pi_2), \;\; \hat{\lambda} = \lambda.$$

Observación 16. En un movimiento ISS la forma en la que se definen las nuevas aplicaciones $\hat{\pi}_1$, $\hat{\pi}_2$, y $\hat{\lambda}$ es independiente de la solución inicial S considerada.

Representación conjuntista del entorno

El entorno ISS de una solución del DTSPMS es el conjunto formado por todas las soluciones que pueden obtenerse a partir de ella realizando un movimiento de Intercambio Dentro de una Pila. En la proposición 17 se detalla cómo construir este tipo de entornos.

Proposición 17. *Sea* $S = (\pi_1, \pi_2, \lambda) \in X$ *una solución factible del DTSPMS y sea* X^λ *el conjunto definido en* (4.4)*. El entorno Intercambio Dentro de una Pila de la solución* S*, denotado por* $\mathrm{ISS}(S)$*, es el conjunto*

$$\mathrm{ISS}(S) = \left\{ (\hat{\pi}_1, \hat{\pi}_2, \hat{\lambda}) \in X^\lambda \,\middle|\, \exists u, v \in D \text{ s.a. } \hat{\pi}_1 = \mathcal{I}_{i_1}^{j_1}(\pi_1),\ \hat{\pi}_2 = \mathcal{I}_{i_2}^{j_2}(\pi_2) \right\}$$

donde $i_1 = \pi_1^{-1}(u)$, $i_2 = \pi_2^{-1}(u)$, $j_1 = \pi_1^{-1}(v)$, $j_2 = \pi_2^{-1}(v)$.

Tamaño del entorno

El tamaño de un entorno de Intercambio Dentro de una Pila es cuadrático con respecto al número n de encargos del problema, según se detalla a continuación.

Proposición 18. *Sea* $S \in X$ *una solución factible del DTSPMS. El entorno* $\mathrm{ISS}(S)$ *es de tamaño* $O\left(\frac{n^2}{m}\right)$ *en media y de tamaño* $O\left(mQ^2\right)$ *en el caso peor.*

Demostración

Un movimiento ISS sobre una solución $S = (\pi_1, \pi_2, \lambda)$ viene determinado por la pareja de encargos $u, v \in D$ asignados a una misma pila cuyas posiciones se intercambian. El número de parejas de este tipo, al igual que ocurría con los movimientos CS, depende del número de clientes asignados a cada pila en la solución S.

Para cada $k \in P$ sea $n_k = |\lambda^{-1}(k)|$ el tamaño de la pila k en la solución S, es decir, el número de encargos asignados a la pila k. De esta manera, hay $\binom{n_k}{2}$ parejas de encargos asignados a la misma pila k, para cada $k \in P$. Por tanto, hay $\sum_{k=1}^{m} \binom{n_k}{2}$ parejas de encargos asignados a una misma pila, y así el entorno $\mathrm{ISS}(S)$ es de tamaño $O\left(\sum_{k=1}^{m} \binom{n_k}{2}\right)$. Como en media se tiene que $n_k \approx \frac{n}{m}$, el tamaño medio del entorno considerado es

$$O\left(\sum_{k=1}^{m} \binom{n_k}{2}\right) = O\left(m\binom{\frac{n}{m}}{2}\right) = O\left(m\left(\frac{n}{m}\right)^2\right) = O\left(\frac{n^2}{m}\right).$$

Por otro lado, como Q es una cota superior de n_k para cada $k \in P$, en el caso peor el tamaño del entorno $\mathrm{ISS}(S)$ es

$$O\left(\sum_{k=1}^{m} \binom{Q}{2}\right) = O\left(m\binom{Q}{2}\right) = O\left(mQ^2\right)$$

como queríamos demostrar. □

Observación 17. El entorno CS(S) está formado por las soluciones obtenidas a partir de S intercambiando las posiciones de dos encargos asignados a pilas *distintas*, mientras que el entorno ISS(S) está formado por las soluciones obtenidas intercambiando dos encargos asignados a la *misma* pila. El tamaño de la unión de ambos entornos, que son disjuntos, será entonces

$$\left|\mathrm{CS}(S) \cup \mathrm{ISS}(S)\right| = \left|\mathrm{CS}(S)\right| + \left|\mathrm{ISS}(S)\right| = \binom{n}{2} - \sum_{k=1}^{m} \binom{n_k}{2} + \sum_{k=1}^{m} \binom{n_k}{2} = \binom{n}{2}$$

o lo que es lo mismo, el número total de parejas de encargos, que también es $O(n^2)$.

4.3.4. Reinserción

El entorno *Reinserción* (*Reinsertion*, R) de una solución S del DTSPMS, denotado por R(S), está formado por todas aquellas soluciones que se pueden obtener a partir de S eligiendo un encargo e insertándolo en otra posición distinta, tanto en las rutas como en las pilas. Al realizar un movimiento de este tipo, denominado por extensión movimiento de Reinserción (R), se modifican al mismo tiempo las rutas y la asignación de pilas de la solución de partida.

Un movimiento de Reinserción queda definido por:

1. El encargo $u \in D$ que se va a recolocar.
2. La pila $k \in P$ a la que se va a reasignar el encargo u.
3. La posición de la ruta 1 en la que se va a recoger el encargo u.
4. La posición de la ruta 2 en la que se va a entregar el encargo u.

Estos elementos deben elegirse convenientemente para que el movimiento pueda realizarse de forma correcta.

Factibilidad de la solución

La factibilidad de la solución obtenida tras la realización de un movimiento de Reinserción depende de la elección de los elementos que lo determinan. Elegidos $u \in D$, $k \in P$ e i^*, $j^* \in I$, el movimiento consistiría en reasignar el encargo u a la pila k y moverlo a la posición i^* en la ruta 1 y a la j^* en la ruta 2. El encargo u se puede elegir libremente, pero la pila k a la que se reasigna debe tener espacio libre para albergarlo y las posiciones i^* y

j^* deben elegirse de manera que al recolocarse u en las dos rutas del problema ocupando dichas posiciones se respeten las condiciones de precedencia.

En el teorema 2 se detalla cómo elegir adecuadamente los elementos de un movimiento de Reinserción para asegurar la factibilidad de la solución resultante.

Teorema 2. *Sea $S = (\pi_1, \pi_2, \lambda)$ una solución del* DTSPMS *y sean $u \in D$, $k \in P$, i^*, $j^* \in I$ los elementos de un movimiento de Reinserción a realizar sobre S. La solución S' obtenida a partir de S reasignando el encargo u a la pila k y recolocándolo en la posición i^* de la ruta 1 y en la j^* de la ruta 2 es factible si se cumplen las dos condiciones siguientes:*

$$i)\ Si\ \lambda(u) \neq k,\ |\lambda^{-1}(k)| < Q \qquad\qquad ii)\ j^* \in I_2[u, i^*, k] \tag{4.5}$$

donde

$$I_2[u, i^*, k] = \begin{cases} \left(i_2^+, i_2^-\right), & si\ i_2^u \in \left(i_2^+, i_2^-\right) \\ \left[i_2^+, i_2^-\right], & si\ i_2^u \notin \left(i_2^+, i_2^-\right) \end{cases} \tag{4.6}$$

$i_2^u = \pi_2^{-1}(u)$, $i_2^- = \pi_2^{-1}(u^-)$, $i_2^+ = \pi_2^{-1}(u^+)$ y u^- y u^+ son, respectivamente, los encargos asignados a la pila k inmediatamente antes y después que u tras ser reinsertado en la posición i^* de la ruta 1.

Demostración

La condición i) es necesaria para asegurar que, si el encargo u se cambia de pila ($\lambda(u) \neq k$), la nueva pila a la que se asigna tenga espacio libre y no exceda la capacidad máxima permitida Q al añadírsele otro encargo.

Sean $i_1^u = \pi_1^{-1}(u)$ la posición del encargo u en la ruta 1 de S y $v = \pi_1(i^*)$ el encargo recogido en la posición i^* de la ruta 1. Al reinsertar u en la posición i^* de la ruta 1 se presentan tres casos:

- $i_1^u < i^*$: La nueva posición que ocupará el encargo u en la ruta 1 es *posterior* a la que ocupaba, es decir, u es visitado antes que v en la ruta 1 de S. Ello provoca que, al recolocar u en la posición i^*, el encargo v se vea desplazado a la posición $i^* - 1$ de la ruta 1, al igual que todos los encargos situados en las posiciones $\{i_1^u + 1, \cdots, i^*\}$, que se retrasan una posición. Así, para cada $j \in \{i_1^u + 1, \cdots, i^*\}$, el encargo que ocupaba la posición j pasará a ocupar la posición $j - 1$ en la ruta 1, mientras que el encargo u, que ocupaba la posición i_1^u, pasará a ocupar la posición i^*.
- $i_1^u = i^*$: La nueva posición del encargo u coincide con la antigua, por lo que la ruta 1 no se modifica.
- $i_1^u > i^*$: La nueva posición que ocupará el encargo u en la ruta 1 es *anterior* a la que ocupaba, es decir, u es visitado después que v en la ruta 1 de S. Ello provoca que el

encargo v sea desplazado a la posición i^*+1 de la ruta 1 para acomodar el encargo u en la posición i^*, al igual que les ocurre a todos los encargos situados en las posiciones $\{i^*, \cdots, i_1^u - 1\}$, que se adelantan una posición. Así, para cada $j \in \{i^*, \cdots, i_1^u - 1\}$, el encargo que ocupaba la posición j pasará a ocupar la posición $j+1$ en la ruta 1, mientras que el encargo u, que ocupaba la posición i_1^u, pasará a ocupar la posición i^*.

El análisis precedente muestra que para realizar la reinserción del encargo u en la ruta 1 hay que mover los encargos recogidos entre la antigua posición i_1^u de u y su nueva posición i^*, ganándose o perdiéndose una posición dependiendo del orden relativo que ocupen i_1^u e i^*.

El conjunto de posiciones posibles a las que se puede mover el encargo u en la ruta 2 manteniendo la factibilidad de la solución es un intervalo $(i_2^+, i_2^-) \subset I$ que depende de la posición i^* de reinserción elegida para la ruta 1 y de la pila k a la que se asigna u. Los extremos i_2^-, i_2^+ de dicho intervalo son las posiciones en la ruta 2 de los encargos asignados a la pila k visitados inmediatamente antes y después del encargo u en la nueva ruta 1. Estos dos encargos se denotarán, respectivamente, por u^- y u^+, y sus posiciones en la ruta 1 de S por $i_1^- = \pi_1^{-1}(u^-), i_1^+ = \pi_1^{-1}(u^+)$, respectivamente. La definición de los encargos u^- y u^+, o equivalentemente las posiciones i_1^-, i_1^+, en los tres casos anteriores es:

- $\underline{i_1^u < i^*}$: El encargo v, que ocupa la posición i^* en la ruta 1, va a pasar a la posición i^*-1, y por tanto será visitado antes que u, que ocupará la posición i^*. Así, u^- es el último encargo distinto de u y asignado a la pila k que sea recogido en la ruta 1 de S en una posición anterior *o igual* a i^*, mientras que u^+ es el primer encargo asignado a la pila k recogido en la ruta 1 de S en una posición posterior a i^*.

- $\underline{i_1^u = i^*}$: Se tiene que $u = v$ e $i^* = i_1^u$, por lo que u^- es el último encargo asignado a la pila k recogido en la ruta 1 de S en una posición anterior a i^* y u^+ es el primer encargo asignado a la pila k recogido en la ruta 1 de S en una posición posterior a i^*.

- $\underline{i_1^u > i^*}$: El encargo v, que ocupa la posición i^* en la ruta 1, va a pasar a la posición i^*+1, y por tanto será visitado después que u, que ocupará la posición i^*. Así, u^- es el último encargo asignado a la pila k recogido en la ruta 1 de S en una posición anterior a i^*, mientras que u^+ es el primer encargo distinto de u y asignado a la pila k que sea recogido en la ruta 1 de S en una posición posterior *o igual* a i^*.

Si no existe ningún encargo u^+ con las características exigidas anteriormente es porque al cambiar de posición en la ruta 1 el encargo u ha pasado a ser el último encargo de la pila k; en ese caso, u^+ será el primer encargo de la ruta 2 de S, esto es $i_2^+ = 1$, pues u

podría ocupar cualquier posición anterior a i_2^- en la ruta 2. Análogamente, si no existe u^- en las condiciones anteriores es porque el encargo u ha pasado a ser el primer encargo de la pila k en la ruta 1; entonces, u^- será el último encargo de la ruta 2 de S, esto es $i_2^- = n$, pues u podría ocupar cualquier posición posterior a i_2^+ en la ruta 2 .

Al calcular los encargos $u^-, u^+ \in D$, que dependen de i^*, u, y k, se obtienen i_2^- e i_2^+, que son las posiciones de u^- y u^+ en la ruta 2 de S, respectivamente. Estas posiciones siempre verifican que $i_2^+ < i_2^-$. Hay dos situaciones posibles:

1. $i_2^+ = 1$ ó $i_2^- = n$: En ambos casos es obvio que $i_2^+ < i_2^-$.
2. Caso general: u^- se recoge antes que u^+ en la ruta 1 de S y ambos encargos están asignados a la misma pila k. Entonces, según la regla LIFO que rige cada pila, u^+ se entrega antes que u^- en la ruta 2, y por tanto $i_2^+ < i_2^-$.

El encargo u, por tanto, siempre se puede recolocar en cualquier posición $j \in \left(i_2^+, i_2^-\right)$ de la ruta 2, ya que entre ellas no hay ningún encargo asignado a la pila k y están dentro de las posiciones límite marcadas por los clientes anterior y posterior a u en la pila k.

Los extremos del intervalo serán posiciones válidas o no dependiendo de la posición del encargo u en la ruta 2 de S. Sea $i_2^u = \pi_2^{-1}(u) \in I$ dicha posición. Si $i_2^u \in \left(i_2^+, i_2^-\right)$, al recolocarse u en la posición i_2^+ de la ruta 2 el encargo u^+ pasaría a entregarse inmediatamente después que u, mientras que al recolocarse en la posición i_2^- el encargo u^- pasaría a entregarse inmediatamente antes que u. En ambos casos se violan las relaciones de precedencia de la pila k según las cuales u tiene que entregarse antes que u^+ y después que u^- en la ruta 2, obteniéndose soluciones no factibles. Por el contrario, si i_2^u está fuera del intervalo $\left(i_2^+, i_2^-\right)$, los encargos u^+ y u^- se trasladan correctamente sin dar lugar a ningún tipo de incompatibilidad.

Así, la posición $j^* \in I$ en la cual colocar el encargo u en la ruta 2 debe pertenecer al intervalo $I_2[u, i^*, k]$, definido como:

$$I_2[u, i^*, k] = \begin{cases} \left(i_2^+, i_2^-\right), & \text{si } i_2^u \in \left(i_2^+, i_2^-\right) \\ \left[i_2^+, i_2^-\right], & \text{si } i_2^u \notin \left(i_2^+, i_2^-\right) \end{cases}$$

□

En la Figura 4.4 se ilustra cómo se modificaría una ruta π_1 y la asignación de pilas λ al realizar un movimiento de Reinserción en el que el encargo u asignado a la pila k_1 se reinserta en la posición i^* de la ruta 1 y se reasigna a la pila k_2. Se distinguen dos casos, dependiendo de si u ocupa una posición anterior o posterior a i^* en π_1. El encargo v es

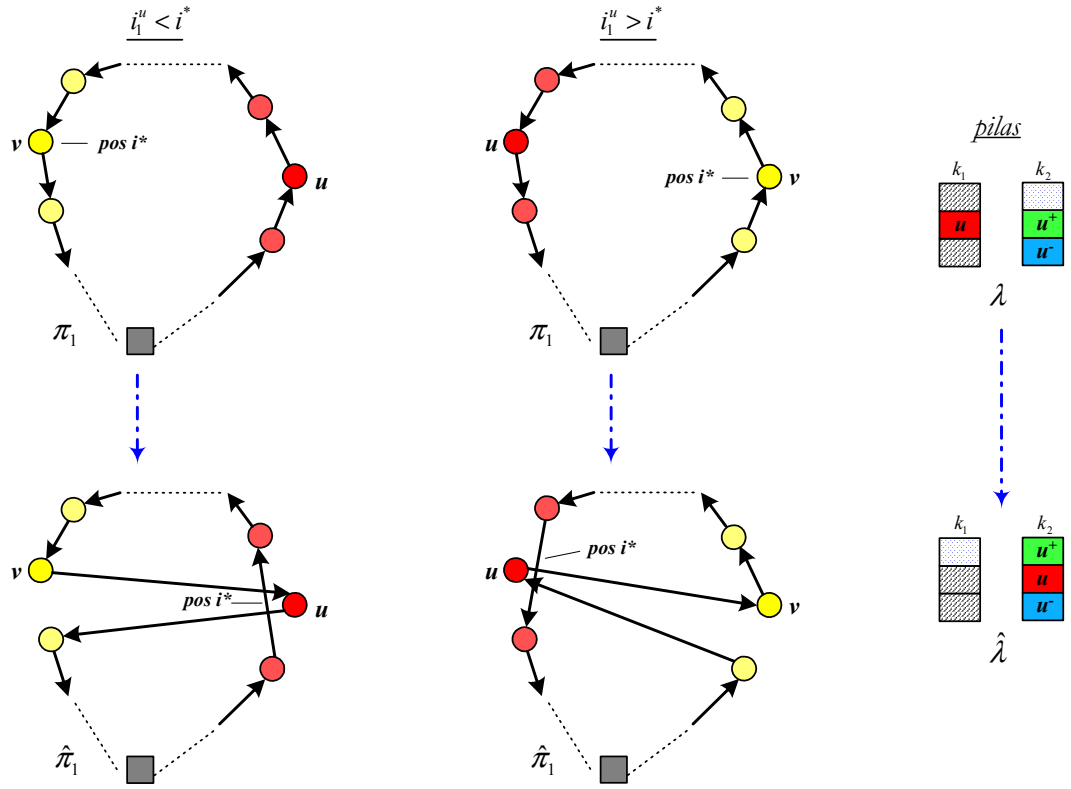


Figura 4.4: Un movimiento de Reinserción

aquel que ocupa la posición i^* en π_1 y u^+ y u^- son los encargos entre los que se sitúa u en la pila k_2 tras el movimiento. La reinserción del encargo u en la ruta 2 se realiza de forma análoga.

Observación 18. En un movimiento de Reinserción, el encargo a ser reinsertado y la posición de reinserción en una de las rutas se pueden elegir libremente. La pila a la que reasignarlo debe ser elegida entre aquellas que tengan espacio libre, mientras que la posición de reinserción en la otra ruta se elige de entre las contenidas en el intervalo definido en 4.6.

Observación 19. El intervalo $I_2[u, i^*, k]$ depende de u, i^* y k, elegidos previamente, y por ello se hacen notar en el nombre. Si se fija primero la posición j^* donde reinsertar u en la ruta 2, el intervalo $I_1[u, j^*, k]$ de posiciones factibles de reinserción en la ruta 1 se define de forma análoga, cambiando los papeles que juegan las rutas 1 y 2 en el proceso anterior.

Observación 20. Si el tamaño máximo Q de las pilas es totalmente ajustado, es decir, $mQ = n$, todas las pilas estarán completamente llenas en cualquier solución factible del problema, de manera que un encargo sólo puede cambiarse de una pila k_1 a otra pila k_2 si a su vez se cambia otro encargo de la pila k_2 a la k_1. En un movimiento de Reinserción

no se realizan intercambios entre dos encargos, sino que se elige uno solo y se recoloca en otra posición; por ello, si el tamaño de las pilas es ajustado, el encargo elegido para ser recolocado no puede cambiarse de pila, ya que en dicha pila no habría espacio disponible a no ser que se moviese otro encargo de los asignados a ella. En caso de que el tamaño de las pilas no sea ajustado, un encargo puede reasignarse a otra pila siempre y cuando ésta disponga de espacio libre, es decir, el número de encargos asignados a ella sea menor que Q.

Realización de un movimiento

Un movimiento de Reinserción queda determinado por un encargo, una pila y una posición en cada ruta. La definición 27 precisa este tipo de movimientos.

Definición 27. Un *movimiento de Reinserción* sobre una solución del DTSPMS, denotado por R, consiste en cambiar un encargo de posición en las dos rutas del problema, reasignándolo a otra pila si fuera necesario. Sean $u \in D$ un encargo, $k \in P$ una pila e i^*, $j^* \in I$ dos posiciones en las rutas. El conjunto X^R_{u,k,i^*,j^*} formado por las soluciones sobre las que se puede realizar el movimiento R que queda determinado por el encargo u, la pila k, la posición i^* en la ruta 1 y la posición j^* de la ruta 2 se define como:

$$X^R_{u,k,i^*,j^*} = \{(\pi_1, \pi_2, \lambda) \in X \text{ sujeta a las condiciones (4.5)}\}.$$

Sobre este conjunto se define el *operador movimiento*

$$\begin{array}{rcl} \mathrm{R}[u,k,i^*,j^*] : X^R_{u,k,i^*,j^*} & \longrightarrow & X \\ S & \longmapsto & \mathrm{R}[u,k,i^*,j^*](S) \equiv \text{solución tras movimiento R} \end{array}$$

que asigna a cada solución S de su dominio la solución que se obtiene realizando sobre S el movimiento de Reinserción en el que el encargo u se reasigna a la pila k y se reinserta en la posición i^* de la ruta 1 y en la posición j^* de la ruta 2.

Definición 28. Sean A y B dos conjuntos finitos con el mismo cardinal, $\alpha \in A$, $\beta \in B$ y $\mathrm{Biy}(A,B) = \{\pi : A \longrightarrow B \mid \pi \text{ es biyectiva}\}$. Se define el operador

$$\begin{array}{rcl} \mathcal{R}^\beta_\alpha : \mathrm{Biy}(A,B) & \longrightarrow & \mathrm{Biy}(A,B) \\ \pi & \longmapsto & \mathcal{R}^\beta_\alpha(\pi) \end{array}$$

que asigna a cada función $\pi \in \mathrm{Biy}(A,B)$ otra función biyectiva $\theta = \mathcal{R}^\beta_\alpha(\pi) \in \mathrm{Biy}(A,B)$ definida de la siguiente forma:

i. Si $\pi^{-1}(\beta) < \alpha$

$$\theta(a) = \begin{cases} \pi(a) & \text{si} \quad a < \pi^{-1}(\beta) \quad \text{ó} \quad a > \alpha \\ \pi(a+1) & \text{si} \quad \pi^{-1}(\beta) \leq a < \alpha \\ \beta & \text{si} \quad a = \alpha \end{cases}$$

ii. Si $\pi^{-1}(\beta) > \alpha$

$$\theta(a) = \begin{cases} \pi(a) & \text{si} \quad a < \alpha \quad \text{ó} \quad a > \pi^{-1}(\beta) \\ \pi(a-1) & \text{si} \quad \alpha < a \leq \pi^{-1}(\beta) \\ \beta & \text{si} \quad a = \alpha \end{cases}$$

iii. Si $\pi^{-1}(\beta) = \alpha$

$$\theta(a) = \pi(a)$$

para cada $a \in A$.

Observación 21. El operador $\mathcal{R}_\alpha^\beta$ introducido en la definición 28 facilita la obtención de rutas construidas reinsertando un elemento en una nueva posición. Si se interpreta A como un conjunto ordenado de índices y B como un conjunto de elementos, una aplicación $\pi \in \text{Biy}(A, B)$ asigna a cada índice un elemento, induciendo un orden en ellos. En el contexto de problemas de rutas, la aplicación π define una ruta que recorre todos los elementos de B. Así, para $\alpha \in A$ y $\beta \in B$, la aplicación $\mathcal{R}_\alpha^\beta(\pi)$ representaría la ruta obtenida a partir de π insertando el elemento β en la posición α y modificando las posiciones de los demás de forma adecuada.

La forma de construir la solución obtenida tras un movimiento de Reinserción viene dada por la proposición 19.

Proposición 19. *La solución $\hat{S} = (\hat{\pi}_1, \hat{\pi}_2, \hat{\lambda}) = R[u, k, i^*, j^*](S)$ resultado de aplicar el operador movimiento $\text{R}[u, k, i^*, j^*]$ sobre una solución $S = (\pi_1, \pi_2, \lambda) \in X^R_{u,k,i^*,j^*}$ es la siguiente:*

$$\hat{\lambda}(w) = \begin{cases} \lambda(w) & si \quad w \in D \backslash \{u\} \\ k & si \quad w = u \end{cases}$$

Definición de $\hat{\pi}_1$. Se distinguen 3 casos:

- *Si $i_1^u < i^*$:*

$$\hat{\pi}_1(i) = \begin{cases} \pi_1(i) & si \quad i < i_1^u \ \acute{o} \ i > i^* \\ \pi_1(i+1) & si \quad i_1^u \leq i < i^* \\ u & si \quad i = i^* \end{cases} \qquad \forall i \in I$$

- *Si* $i_1^u > i^*$*:*

$$\hat{\pi}_1(i) = \begin{cases} \pi_1(i) & si \quad i < i^* \ \acute{o} \ i > i_1^u \\ \pi_1(i-1) & si \quad i^* < i \leq i_1^u \\ u & si \quad i = i^* \end{cases} \qquad i \in I$$

- *Si* $i_1^u = i^*$*:*

$$\hat{\pi}_1(i) = \pi_1(i) \qquad \forall i \in I$$

Definición de $\hat{\pi}_2$*. Se distinguen 3 casos:*

- *Si* $i_2^u < j^*$*:*

$$\hat{\pi}_2(i) = \begin{cases} \pi_2(i) & si \quad i < i_2^u \ \acute{o} \ i > j^* \\ \pi_2(i+1) & si \quad i_2^u \leq i < j^* \\ u & si \quad i = j^* \end{cases} \qquad \forall i \in I$$

- *Si* $i_2^u > j^*$*:*

$$\hat{\pi}_2(i) = \begin{cases} \pi_2(i) & si \quad i < j^* \ \acute{o} \ i > i_2^u \\ \pi_2(i-1) & si \quad j^* < i \leq i_2^u \\ u & si \quad i = j^* \end{cases} \qquad \forall i \in I$$

- *Si* $i_2^u = j^*$*:*

$$\hat{\pi}_2(i) = \pi_2(i) \qquad \forall i \in I$$

donde $i_1^u = \pi_1^{-1}(u)$ *e* $i_2^u = \pi_2^{-1}(u)$ *son las posiciones de* u *en las rutas 1 y 2, respectivamente.*

Observación 22. De forma equivalente, utilizando el operador introducido en la definición 28, las rutas de la solución $\hat{S} = (\hat{\pi}_1, \hat{\pi}_2, \hat{\lambda})$ se pueden definir abreviadamente como

$$\hat{\pi}_1 = \mathcal{R}_{i^*}^u(\pi_1), \qquad \hat{\pi}_2 = \mathcal{R}_{j^*}^u(\pi_2).$$

En el Ejemplo 1 se presenta un pequeño ejemplo con 7 encargos y 3 pilas de capacidad 3 ($D = \{1, \cdots, 7\}$, $m = 3$, $Q = 3$) para ilustrar la proposición 19.

Ejemplo 1. Consideremos la solución $S = (\pi_1, \pi_2, \lambda)$ definida de la siguiente manera:

$$\begin{array}{ll} \pi_1(i) = i & \forall i \in D \\ \pi_2(i) = 8 - i & \forall i \in D \end{array} \qquad \lambda(c) = \begin{cases} 1 & \text{si } c \in \{1, 2, 5\} \\ 2 & \text{si } c \in \{3, 7\} \\ 3 & \text{si } c \in \{4, 6\} \end{cases}$$

La ruta de recogida es [0-1-2-3-4-5-6-7-0] y la de entrega la opuesta, mientras que las pilas 2 y 3 tienen un hueco libre para albergar un nuevo encargo. La forma de construir la solución

$\hat{S} = (\hat{\pi}_1, \hat{\pi}_2, \hat{\lambda}) = \mathrm{R}[6,2,4,4](S)$ siguiendo la proposición 19 se detalla a continuación. El encargo que entra en juego es $u = 6$, que se moverá de la pila 3 a la 2 y se reinsertará en las posiciones $i^* = 4$ y $j^* = 4$. Así, $i_u^1 = \pi_1^{-1}(6) = 6 > i^* = 4$, por lo que la nueva ruta de recogida será

$$\hat{\pi}_1(i) = \begin{cases} \pi_1(i) & \text{si} \quad i = 1,2,3,7 \\ \pi_1(i-1) & \text{si} \quad i = 5,6 \\ 6 & \text{si} \quad i = 4 \end{cases}$$

que es la ruta [0-1-2-3-6-4-5-7-0]. De forma análoga se construye la nueva ruta de entrega, teniendo en cuenta que $i_u^2 = \pi_2^{-1}(6) = 2 < j^* = 4$,

$$\hat{\pi}_2(i) = \begin{cases} \pi_2(i) & \text{si} \quad i < 2 \ \text{ó} \ i > 4 \\ \pi_2(i+1) & \text{si} \quad i = 2,3 \\ 6 & \text{si} \quad i = 4 \end{cases}$$

que es la ruta [0-7-5-4-6-3-2-1-0]. La nueva asignación de pilas es

$$\lambda(c) = \begin{cases} 1 & \text{si } c \in \{1,2,5\} \\ 2 & \text{si } c \in \{3,7,6\} \\ 3 & \text{si } c \in \{4\} \end{cases}$$

y se comprueba fácilmente que la nueva solución es la que corresponde al movimiento de Reinserción requerido y es factible.

Representación conjuntista del entorno

El entorno R de una solución del DTSPMS es el conjunto formado por todas las soluciones que pueden obtenerse a partir de ella realizando un movimiento de Reinserción. La proposición 20 detalla cómo construir este tipo de entornos.

Proposición 20. *Sea* $S = (\pi_1, \pi_2, \lambda) \in X$ *una solución factible del* DTSPMS. *El entorno Reinserción de la solución* S*, denotado por* $R(S)$*, es el conjunto*

$$\mathrm{R}(S) = \left\{ (\hat{\pi}_1, \hat{\pi}_2, \hat{\lambda}) \in X \,\middle|\, \textit{existen } u \in D,\ k \in P,\ i^* \in I,\ j^* \in I_2[u, i^*, k] \ \textit{sujeto a } C_\lambda,\ C_\pi \right\}$$

donde las condiciones C_λ *y* C_π *son:*

$$C_\lambda : \left|\lambda^{-1}(k)\right| < Q \ \textit{si}\ \lambda(u) \neq k,\ \hat{\lambda}(u) = k \ \textit{y}\ \hat{\lambda}(w) = \lambda(w),\ \forall w \in D \setminus \{u\} \tag{4.7}$$

$$C_\pi : \ \hat{\pi}_1 = \mathcal{R}_{i^*}^u(\pi_1), \quad \hat{\pi}_2 = \mathcal{R}_{j^*}^u(\pi_2) \tag{4.8}$$

Demostración

La condición C_λ asegura que si el encargo u se cambia de pila, la pila k elegida tenga espacio libre en S. Además, también exige que la asignación de pilas de $\hat{S}$ sea idéntica a la de S salvo para el encargo u, al que se le asigna la pila k.

En la condición C_π se pide que las rutas de $\hat{S}$ se obtengan a partir de las de S recolocando el encargo u en las posiciones i^* y j^* de las rutas 1 y 2, respectivamente.

Por último, $j^* \in I_2[u, i^*, k]$ garantiza que $\hat{S}$ respete las condiciones de precedencia. □

Observación 23. El conjunto $\mathrm{R}(S)$ contiene todas aquellas soluciones que se pueden obtener a partir de S mediante un movimiento de Reinserción fijando primero la posición i^* de la ruta 1 y eligiendo posteriormente la posición j^* entre las posiciones factibles de la ruta 2. Es inmediato comprobar, sin más que cambiar los papeles que juegan las rutas 1 y 2 en el problema, que las soluciones que se obtienen fijando primero la posición j^* de la ruta 2 y eligiendo i^* entre las posiciones factibles que quedan en la ruta 1 dan lugar al mismo conjunto $\mathrm{R}(S)$, es decir,

$$R(S) = \left\{ (\hat{\pi}_1, \hat{\pi}_2, \hat{\lambda}) \in X \;\middle|\; \text{existen } u \in D,\, k \in P,\, j^* \in I,\, i^* \in I_1[u, j^*, k] \text{ sujeto a } C_\lambda,\, C_\pi \right\}$$

donde las condiciones C_λ y C_π son las definidas en (4.7) y (4.8).

Tamaño del entorno

El tamaño de un entorno de Reinserción es cuadrático con respecto al producto nm del número de encargos del problema y el número de pilas disponibles, según detalla la proposición 21.

Proposición 21. *Sea $S \in X$ una solución factible del DTSPMS. El entorno $\mathrm{R}(S)$ es de tamaño $O(n^2m^2)$ en media y de tamaño $O(n^3m)$ en el caso peor.*

Demostración

Un movimiento de Reinserción sobre una solución inicial S viene unívocamente determinado por la 4-upla (u, k, i^*, j^*), donde $u \in D$ es el encargo que se va a recolocar, $k \in P$ la pila a la que se va a reasignar e $i^* \in I$, $j^* \in I_2[u, i^*, k]$ las posiciones donde recolocarlo en las rutas 1 y 2, respectivamente. Cada 4-upla da lugar a un movimiento distinto, por lo que el número de movimientos posibles coincide con el número de 4-uplas de la forma (u, k, i^*, j^*) que dan lugar a un movimiento factible.

Se tiene que $|D| = n$, $|P| = m$ y $|I| = n$. El cardinal de $I_2[u, i^*, k]$ no puede calcularse de forma exacta a priori, ya que depende de i^* y de las características de la solución

inicial S. Dicho cardinal, que es la longitud de un intervalo de la forma $\left[i_2^+, i_2^-\right]$, donde i_2^+ e i_2^- son las posiciones de dos clientes consecutivos en una pila, coincide, en media, con la separación media en las rutas de dos clientes asignados consecutivamente a la misma pila. Esta separación media es el número de pilas menos uno $(m-1)$, que es el número esperado de encargos situados en una ruta entre dos encargos consecutivos de una pila. Así, el tamaño medio del entorno es

$$\left|\left\{ (u,k,i^*,j^*) \,|\, u \in D,\ k \in P,\ i \in I,\ j^* \in I_2[u,i^*,k] \right\}\right| =$$

$$= |D| \cdot |P| \cdot |I| \cdot \left|I_2[u,i^*,k]\right| = O\big(n \cdot m \cdot n \cdot (m-1)\big) = O(n^2m^2).$$

Por otro lado, el cardinal de $I_2[u,i^*,k]$ está acotado por n, por lo que el tamaño del entorno en el caso peor es $|D| \cdot |P| \cdot |I| \cdot \left|I_2[u,i^*,k]\right| = O\big(n \cdot m \cdot n \cdot n\big) = O(n^3m)$. □

Observación 24. Si el tamaño máximo Q de las pilas es totalmente ajustado, es decir, $mQ = n$, en los movimientos de Reinserción el encargo recolocado no se puede cambiar de pila. Así, si $S = (\pi_1, \pi_2, \lambda)$, el tamaño medio del entorno sería

$$\left|\left\{ (u,\lambda(u),i^*,j^*) \,|\, u \in D,\ i \in I,\ j^* \in I_2[i^*,u] \right\}\right| = |D| \cdot |I| \cdot \left|I_2[i^*,u]\right| = O(n \cdot n \cdot m) = O(n^2m).$$

4.3.5. r-Permutación en Ruta

El entorno *r-Permutación en Ruta* (*r-Route Permutation*, r-RP) de una solución S del DTSPMS, denotado por r-RP(S), está formado por todas aquellas soluciones que se pueden obtener a partir de S permutando r encargos visitados de forma consecutiva en alguna de las rutas de la solución y asignados a distintas pilas dos a dos. Al realizar un movimiento de este tipo, denominado por extensión movimiento r-Permutación en Ruta (r-RP), sólo se modifica la parte de la solución relativa a la ruta elegida para la realización del movimiento, manteniéndose intacta la asignación de pilas.

Factibilidad de la solución

Un movimiento r-RP puede realizarse tanto en la ruta de entrega como en la ruta de recogida, no siendo necesario actualizar la ruta sobre la que no actúe el movimiento, como prueba la proposición 22.

Proposición 22. *Sea $u \in D$ el primero de los r encargos asignados a pilas distintas y consecutivos en la ruta 1 que se van a permutar en un movimiento r-RP y sea $S = (\pi_1, \pi_2, \lambda)$ la solución inicial sobre la que se realiza el movimiento. La solución S' obtenida*

a partir de S realizando una permutación de las posiciones en la ruta 1 de estos r encargos es factible.

Demostración

Como los r encargos cuyas posiciones en la ruta 1 se van a permutar son *consecutivos* y están asignados a *pilas diferentes*, al ser permutados no cambia la asignación de pilas ni el *orden* en el que los encargos son almacenados en dichas pilas. Al no modificarse la asignación de pilas ni el orden relativo de los encargos en ellas, no es necesario realizar modificación alguna en la ruta de entrega (ruta 2) para mantener la factibilidad de la solución, ya que ésta sigue siendo compatible con la nueva ruta 1. □

Observación 25. El movimiento r-RP sobre la ruta 2 es análogo.

Observación 26. Es claro que para poder realizar un movimiento de este tipo, el número de encargos que se permutan debe ser menor o igual que el número de pilas disponibles, ya que tienen que estar asignados a pilas distintas dos a dos. Por tanto, es necesario que $r \leq m$.

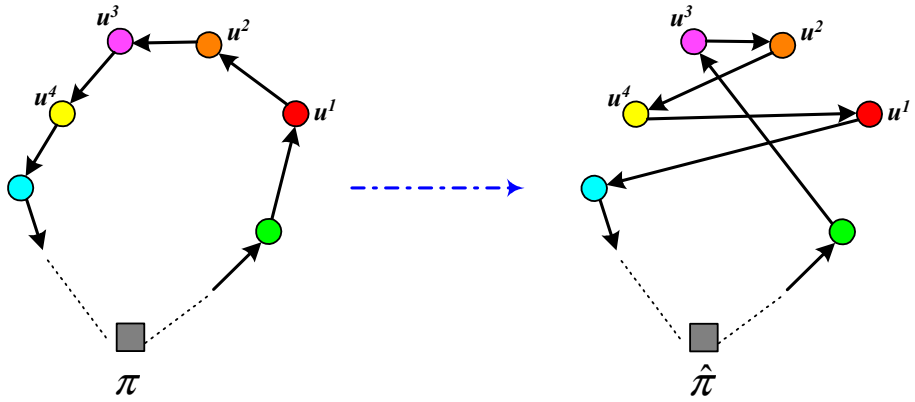


Figura 4.5: Un movimiento de r-Permutación en Ruta

En la Figura 4.5 se ilustra cómo se realiza un movimiento 4-RP sobre una ruta cualquiera en el que se permutan cuatro encargos u^1, u^2, u^3, u^4 consecutivos en r y asignados a pilas distintas. La permutación llevada a cabo en la figura es $(3, 2, 4, 1)$.

Realización de un movimiento

Un movimiento r-RP queda determinado por la ruta sobre la que se aplica, el primero de los r encargos elegidos y la permutación que se va a realizar sobre ellos. La definición 29 precisa este tipo de movimientos.

Definición 29. Un *movimiento r-Permutación en Ruta* sobre una solución del DTSPMS, denotado por r-RP, consiste en realizar una permutación de las posiciones de r encargos sobre una de las rutas del problema; dichos encargos deben ser consecutivos en la ruta elegida y estar asignados a pilas distintas dos a dos. Sean $\delta \in \{1,2\}$ una de las rutas del problema, $u^1 \in D$ un encargo y σ una permutación de r elementos. El conjunto $X^{rRP}_{\delta,u^1,\sigma}$ formado por las soluciones sobre las que se puede realizar el movimiento r-RP que queda determinado por δ, u^1 y σ se define como:

$$X^{rRP}_{\delta,u^1,\sigma} = \{(\pi_1,\pi_2,\lambda) \in X \,|\, i_1 \leq n-r+1,\ \lambda(\pi_\delta(i)) \neq \lambda(\pi_\delta(j))\ \forall i,j \in A,\ i \neq j\}\,.$$

donde $i_1 = \pi_\delta^{-1}(u^1)$ y $A = \{i_1,\ i_1+1,\cdots,\ i_1+r-1\}$. Sobre este conjunto se define el *operador movimiento*

$$\begin{array}{rcl} r\text{-RP}[\delta,u^1,\sigma] : X^{rRP}_{\delta,u^1,\sigma} & \longrightarrow & X \\ S & \longmapsto & r\text{-RP}[\delta,u^1,\sigma](S) \equiv \text{solución tras movimiento } r\text{-RP} \end{array}$$

que asigna a cada solución $S = (\pi_1,\pi_2,\lambda)$ de su dominio la solución que se obtiene realizando sobre S el movimiento de r-Permutación en Ruta en el que se permutan según σ las posiciones $i_1, i_1+1,\cdots,i_1+r-1$ de la ruta δ, donde $i_1 = \pi_\delta^{-1}(u^1)$.

La forma de construir la solución obtenida tras un movimiento r-RP viene dada por la proposición 23.

Proposición 23. *La solución $\hat{S} = (\hat{\pi}_1,\hat{\pi}_2,\hat{\lambda}) = r\text{-RP}[\delta,u^1,\sigma](S)$ resultado de aplicar el operador movimiento r-RP$[\delta,u^1,\sigma]$ sobre una solución $S = (\pi_1,\pi_2,\lambda) \in X^{rRP}_{\delta,u^1,\sigma}$ es la siguiente:*

$$\hat{\pi}_\delta(j) = \begin{cases} \pi_\delta(\sigma^{-1}(j)) & si \quad j \in A \\ \pi_\delta(j) & en\ otro\ caso \end{cases}$$

$$\hat{\pi}_\gamma(i) = \pi_\gamma(i), \quad \forall i \in I \qquad\qquad \hat{\lambda}(u) = \lambda(u), \quad \forall u \in D$$

donde $\gamma \in \{1,2\}\backslash\{\delta\}$, $i_1 = \pi_\delta^{-1}(u^1)$ es la posición en la ruta δ del encargo u^1, $A = \{i_1,\ i_1+1,\cdots,\ i_1+r-1\}$ son las posiciones en la ruta δ de los encargos elegidos para ser permutados y $\sigma : A \longrightarrow A$ representa la permutación que se va a realizar: para cada $j \in A$, el encargo que ocupaba la posición j en la ruta δ pasará a ocupar la posición $\sigma(j)$.

Observación 27. Un movimiento del tipo r-*RP*$[\delta,u^1,\sigma]$ sólo modifica la ruta δ, la otra ruta y la asignación de pilas permanecen inalteradas.

Representación conjuntista del entorno

El entorno r-RP de una solución del DTSPMS es el conjunto formado por todas las soluciones que pueden obtenerse a partir de ella realizando un movimiento de r-Permutación en Ruta. La proposición 24 detalla cómo construir este tipo de entornos.

Proposición 24. *Sean* $S = (\pi_1, \pi_2, \lambda) \in X$ *una solución factible del DTSPMS y*

$$X^{\lambda}(\pi_\delta) = \{(\hat{\pi}_1, \hat{\pi}_2, \hat{\lambda}) \in X | \hat{\lambda} = \lambda,\ \hat{\pi}_\delta = \pi_\delta\} \subset X, \qquad \delta \in \{1, 2\}$$

el conjunto de soluciones factibles con la misma asignación de pilas que S *y la misma ruta* δ*. El entorno r-Permutación en Ruta de la solución* S*, denotado por* r-RP(S)*, es el conjunto*

$$r\text{-RP}(S) = rRP_1 \cup rRP_2$$

donde

$$rRP_\delta = \left\{(\hat{\pi}_1, \hat{\pi}_2, \hat{\lambda}) \in X^{\lambda}(\pi_\gamma) \,\middle|\, \gamma \in \{1,2\}\backslash\{\delta\},\ \exists i_1 \in \{1, \cdots, n+1-r\} \text{ sujeto a } C_1,\ C_2\right\}$$

para $\delta \in \{1, 2\}$*, y las condiciones* C_1 *y* C_2 *son:*

i. $C_1 : \hat{\pi}_\delta(i) = \pi_\delta(i)\ \forall i \in I\backslash\{i_1, \cdots, i_1 + r - 1\}$

ii. $C_2 : \lambda(\hat{\pi}_\delta(i)) \neq \lambda(\hat{\pi}_\delta(j))\ \forall i, j \in \{i_1, \cdots, i_1 + r - 1\},\ i \neq j$

Demostración

En entorno se divide en dos subconjuntos, rRP_1 y rRP_2, cada uno de los cuales contiene las soluciones obtenidas realizando un movimiento r-RP sobre las rutas 1 y 2 de S, respectivamente. Para cada $\delta \in \{1, 2\}$, la condición C_1 indica que la ruta δ de $\hat{S}$ se obtiene a partir de la ruta δ de S permutando r encargos consecutivos. La condición C_2, por último, impone que los r encargos elegidos estén asignados a pilas distintas. □

Tamaño del entorno

El tamaño de un entorno r-RP es lineal sobre el número n de encargos del problema y exponencial sobre el tamaño r de las rachas que se permutan, según muestra la proposición 25.

Proposición 25. *Sea* $S \in X$ *una solución factible del DTSPMS. Entonces, el entorno* r-RP(S) *es a lo sumo de tamaño* $O(nr!)$*.*

Demostración

Elegida la ruta sobre la que realizarlo, un movimiento r-RP viene determinado por el primero de los r encargos que se van a permutar (denotado por u^1), ya que son consecutivos, y por la permutación σ a realizar sobre ellos. El encargo u^1 puede ser cualquiera de los encargos asignados a las $n - r + 1$ primeras posiciones de la ruta elegida, por lo que hay $n - r + 1$ posibilidades. Por otro lado, los r encargos elegidos pueden permutarse de $r!$ formas, por lo que hay $r!$ posibilidades para σ.

Así, sobre cada ruta hay a lo sumo $r!(n - r + 1)$ movimientos posibles del tipo r-RP, por lo que el número total de movimientos es a lo sumo $O(2r!(n - r + 1)) = O(nr!)$. □

Observación 28. En el recuento realizado en la proposición 25 no se ha tenido en cuenta la condición C_2, que impone que los r encargos escogidos estén asignados a pilas distintas. Esto es así porque la asignación de pilas depende de la solución particular que se tenga en cada caso, impidiendo calcular el tamaño exacto del entorno. Así, el tamaño $O(nr!)$ calculado es una cota superior del tamaño real del entorno, pudiendo llegar a ser tamaños muy distintos. Además, la condición C_2 es muy restrictiva, ya que, por lo general, no hay muchas rachas de r clientes consecutivos asignados a pilas distintas. Ello provoca que, en la práctica, en la mayor parte de los casos el tamaño de los entornos r-RP sea sensiblemente menor que $O(nr!)$.

4.3.6. r-Permutación en Pila

El entorno *r-Permutación en Pila* (*r-Stack Permutacion*, r-SP) de una solución S del DTSPMS, denotado por r-SP(S), está formado por todas aquellas soluciones que se pueden obtener a partir de S permutando r encargos almacenados de forma consecutiva en una pila. Al realizar un movimiento de este tipo, denominado por extensión movimiento r-Permutación en Pila y denotado por r-SP, se modifican las posiciones de los encargos elegidos en las dos rutas de la solución, pero no se modifica la asignación de pilas: como consecuencia de la permutación de posiciones en las rutas cambia el orden en el que los encargos permutados se almacenan en la pila, pero éstos siguen en la misma pila en la que estaban.

Factibilidad de la solución

Proposición 26. *Sea $S = (\pi_1, \pi_2, \lambda)$ una solución del* DTSPMS. *La solución S' obtenida a partir de S permutando las posiciones en las rutas 1 y 2 de r encargos asignados consecutivamente a una pila y manteniendo la asignación de pilas intacta, es factible.*

Demostración

Cada pila puede considerarse como un sistema LIFO independiente, de manera que si se realiza cualquier permutación en r encargos asignados a la misma pila k, las relaciones de precedencia se siguen respetando siempre y cuando los cambios se realicen al mismo tiempo en las dos rutas de la solución. Este es el caso del movimiento realizado para obtener S', por lo que dicha solución es factible. □

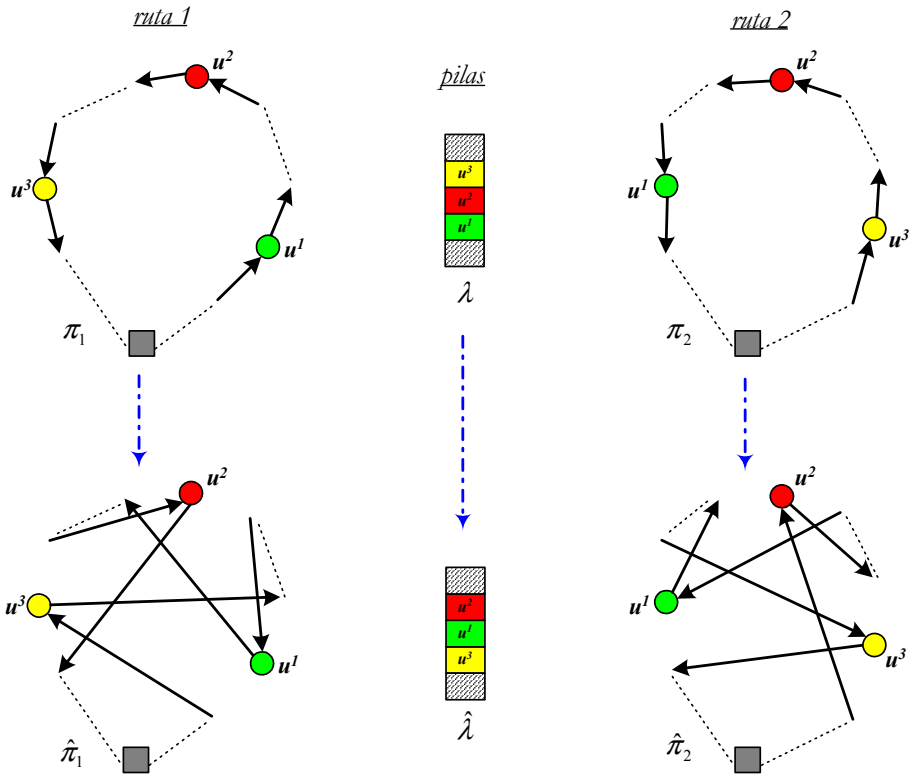


Figura 4.6: Un movimiento de r-Permutación en Pila

La Figura 4.6 ilustra cómo realizar un movimiento 3-SP sobre una solución (π_1, π_2, λ) en la que se permutan 3 encargos u^1, u^2, u^3 asignados consecutivamente a la misma pila. La permutación elegida, $(3, 1, 2)$, se realiza en ambas rutas y en la pila a la que están asignados los encargos u^1, u^2, u^3, obteniéndose una nueva solución $(\hat{\pi}_1, \hat{\pi}_2, \hat{\lambda})$.

Realización de un movimiento

Las condiciones que debe cumplir el conjunto de encargos elegido en un movimiento r-SP se detallan en la proposición 27.

Proposición 27. *Sea* $S = (\pi_1, \pi_2, \lambda)$ *una solución del* DTSPMS *y sea* $u^1 \in D$ *un encargo asignado a la pila* $k \in P$ *detrás del cual se recogen en* S *al menos otros* $r-1$ *encargos asignados a la misma pila* k *que* u^1*. Entonces el conjunto* $U = \{u^2, \cdots, u^r\} \subset D$ *es el conjunto de los* $r-1$ *encargos almacenados de forma consecutiva después de* u^1 *en la pila* k *siguiendo el orden indicado por sus índices (es decir, para cada* $j \in \{1, \cdots, r-1\}$ *el encargo* u^j *se almacena en la pila* k *inmediatamente antes que el* u^{j+1}*)* SI Y SÓLO SI *el conjunto* $\left(U \cup \{u^1\}\right) = \{u^1, \cdots, u^r\}$ *verifica* (4.9) *y cualquiera de las condiciones* (4.10), (4.11).

$$\lambda(u^j) = k,\ \forall j \in \{1, \cdots, r\} \tag{4.9}$$

$$\pi_1^{-1}(u^j) < \pi_1^{-1}(u^{j+1}) \text{ y } \nexists u \in D \big| \lambda(u) = k \text{ y } \pi_1^{-1}(u^j) < \pi_1^{-1}(u) < \pi_1(u^{j+1}),\ \forall j \in \{1, \cdots, r-1\} \tag{4.10}$$

$$\pi_2^{-1}(u^j) > \pi_2^{-1}(u^{j+1}) \text{ y } \nexists u \in D \big| \lambda(u) = k \text{ y } \pi_2^{-1}(u^j) > \pi_2^{-1}(u) > \pi_2(u^{j+1}),\ \forall j \in \{1, \cdots, r-1\} \tag{4.11}$$

Demostración

La condición (4.10) indica que para cada $j \in \{1, \cdots, r-1\}$ el encargo u^j se recoge antes que el u^{j+1} en la pila k y no existe ningún otro encargo asignado a la misma pila que se recoja entre ambos, lo cual implica que el encargo u^j se *recoge inmediatamente antes* que u^{j+1} en la pila k. La condición (4.11) es análoga a la anterior para la otra ruta, asegurando que para cada $j \in \{1, \cdots, r-1\}$ el encargo u^j se *entrega inmediatamente después* que el u^{j+1} en la pila k.

Como S es factible verifica las condiciones de precedencia entre los encargos, de manera que los encargos asignados a una pila concreta son entregados en el orden inverso al que son recogidos. Como consecuencia, las condiciones (4.10) y (4.11) son equivalentes. Así, basta cualquiera de las dos para garantizar la consecutividad de los encargos en la pila k.

$\Rightarrow$*)* Sea $U = \{u^2, \cdots, u^r\} \subset D$ el conjunto de encargos tal que para cada $j \in \{1, \cdots, r-1\}$ el encargo u^j se almacena en la pila k inmediatamente antes que el u^{j+1}. Por la propia definición de U, $\{u^1, \cdots, u^r\}$ verifica (4.9), (4.10) y (4.11).

$\Leftarrow$*)* Supongamos que $\left(U \cup \{u^1\}\right) = \{u^1, \cdots, u^r\}$ verifica (4.9) y cualquiera de las condiciones (4.10), (4.11). La condición (4.9) implica que todos los encargos del conjunto U están asignados a la misma pila k, mientras que cualquiera de las condiciones equivalentes (4.10), (4.11) asegura que los encargos $u^2, \cdots, u^r$ se almacenan de forma consecutiva después de u^1 en la pila k. Así, el conjunto $U = \{u^2, \cdots, u^r\} \subset D$ es el conjunto pedido. □

Un movimiento r-SP queda determinado por el primero de los encargos que se van a permutar (los demás son consecutivos en su misma pila) y la permutación que se va a realizar sobre ellos. La definición 30 precisa este tipo de movimientos.

Definición 30. Un *movimiento r-Permutación en Pila* sobre una solución del DTSPMS, denotado por r-SP, consiste en realizar una permutación de las posiciones de r encargos; los cambios deben realizarse sobre las dos rutas del problema y los encargos elegidos deben estar asignados consecutivamente a una pila. Sean $u^1 \in D$ un encargo del problema y σ una permutación de r elementos. El conjunto $X^{rSP}_{u^1,\sigma}$ formado por las soluciones sobre las que se puede realizar el movimiento r-SP que queda determinado por u^1 y σ se define como:

$$X^{rSP}_{u^1,\sigma} = \left\{(\pi_1,\pi_2,\lambda) \in X \,|\, \exists i_2,\cdots,i_r > \pi_1^{-1}(u^1) \text{ tales que } \lambda(\pi_1(i_j)) = \lambda(u^1)\ \forall j = 2,\cdots,r\right\}.$$

Sobre este conjunto se define el *operador movimiento*

$$\begin{array}{rcl} r\text{-SP}[u^1,\sigma] : X^{rSP}_{u^1,\sigma} & \longrightarrow & X \\ S & \longmapsto & r\text{-SP}[u^1,\sigma](S) \equiv \text{solución tras movimiento } r\text{-SP} \end{array}$$

que asigna a cada solución $S = (\pi_1, \pi_2, \lambda)$ de su dominio la solución que se obtiene realizando sobre S el movimiento de r-Permutación en Pila en el que se permutan en las dos rutas del problema, según σ, las posiciones de los encargos $\{u^1,\cdots,u^r\} \subset D$, los cuales verifican las condiciones (4.9) y (4.10).

La forma de construir la solución que se obtiene tras un movimiento r-SP viene dada por la proposición 28.

Proposición 28. *La solución $\hat{S} = (\hat{\pi}_1, \hat{\pi}_2, \hat{\lambda}) = r\text{-SP}[u^1,\sigma](S)$ resultado de aplicar el operador movimiento r-SP$[u^1,\sigma]$ sobre una solución $S = (\pi_1,\pi_2,\lambda) \in X^{rSP}_{u^1,\sigma}$ es la siguiente:*

$$\hat{\pi}_\delta(i) = \begin{cases} \pi_\delta\left(i_\delta^{\sigma^{-1}(j)}\right) & \text{si } \exists j \in A \text{ tal que } i = i_\delta^j \\ \pi_\delta(i) & \text{en otro caso} \end{cases} \qquad \forall i \in I,\ \forall \delta \in \{1,2\}$$

$$\hat{\lambda}(u) = \lambda(u), \quad \forall u \in D$$

donde $\{u^1,\cdots,u^r\} \subset D$ es el único conjunto de encargos que verifica (4.9) y (4.10), $A = \{1,\cdots,r\}$, $i_\delta^j = \pi_\delta(u^j)$ es la posición del encargo u^j en la ruta δ de S, $\forall j \in A$, $\forall \delta \in \{1,2\}$, y $\sigma : A \longrightarrow A$ es la permutación de los encargos $u_1,\cdots,u_r \in D$ que se va a realizar: $\forall j \in A$, $\forall \delta \in \{1,2\}$, el encargo u^j pasaría de ocupar la posición i_δ^j en la ruta δ de S a ocupar la posición $i_\delta^{\sigma(j)}$ en la ruta δ de $\hat{S}$, o equivalentemente, el encargo que ocuparía la posición i_δ^j en la ruta δ de $\hat{S}$ sería $u^{\sigma^{-1}(j)} = \pi_\delta\left(i_\delta^{\sigma^{-1}(j)}\right)$.

Observación 29. Se realiza la misma permutación en ambas rutas, manteniéndose intacta la asignación de pilas.

Representación conjuntista del entorno

El entorno r-SP de una solución del DTSPMS es el conjunto formado por todas las soluciones que pueden obtenerse a partir de ella realizando un movimiento de r-Permutación en Pila. La proposición 29 detalla cómo construir este tipo de entornos.

Proposición 29. *Sean $S = (\pi_1, \pi_2, \lambda) \in X$ una solución factible del DTSPMS , $A = \{1, \cdots, r\}$, Biy(A) el conjunto de las permutaciones de A y X^λ el conjunto definido en* (4.4). *El entorno r-Permutación en Pila de la solución S, denotado por r-SP(S), es el conjunto*

$$\text{r-SP}(S) = \left\{ (\hat{\pi}_1, \hat{\pi}_2, \hat{\lambda}) \in X^\lambda) \,\middle|\, \exists \sigma \in Biy(A),\, \exists u^1, \cdots, u^r \in D \text{ sujetos a } C_1,\, C_2,\, C_3,\, C_4 \right\}$$

donde las condiciones C_i, $i = 1, 2, 3, 4$ son

i. $C_1 : \exists k \in P$ *tal que* $\lambda(u^j) = k,\ \forall j \in A$

ii. $C_2 : \pi_1^{-1}(u^j) < \pi_1^{-1}(u^{j+1})$ *y* $\nexists u \in D \big| \lambda(u) = k$ *y* $\pi_1^{-1}(u^j) < \pi_1^{-1}(u) < \pi_1^{-1}(u^{j+1}),\ \forall j \in A \backslash \{r\}$

iii. $C_3 : \hat{\pi}_\delta\big(i_\delta^j\big) = \pi_\delta\big(i_\delta^{\sigma^{-1}(j)}\big)\ \ \forall j \in A,\ \forall \delta \in \{1, 2\}$

iv. $C_4 : \hat{\pi}_\delta(i) = \pi_\delta(i)\ \ \forall i \in I \backslash \left\{ i_\delta^1, \cdots, i_\delta^r \right\},\ \forall \delta \in \{1, 2\}$

$i_\delta^j = \pi_\delta^{-1}(u^j)$ *la posición del encargo u^j en la ruta δ de S, $\forall j \in A$ y $\forall \delta \in \{1, 2\}$.*

Demostración

La condición C_1 exige que los encargos $\left\{ u^1, \cdots, u^r \right\}$ que se van a permutar estén asignados a la misma pila, mientras que la condición C_2 asegura que estén almacenados consecutivamente en dicha pila. La condición C_3 indica cómo realizar la permutación de los encargos elegidos en las dos rutas de la solución, y por último la C_4 garantiza que los encargos no permutados se mantengan en la misma posición. □

Tamaño del entorno

El tamaño de un entorno r-SP es lineal sobre el número n de encargos del problema y exponencial sobre el tamaño r de las rachas que se permutan, según muestra la proposición 30.

Proposición 30. *Sea $S \in X$ una solución factible del* DTSPMS. *El entorno r-SP(S) es de tamaño $O(nr!)$ en media y de tamaño $O(m(Q - r + 1)r!)$ en el caso peor.*

Demostración

Hay m pilas disponibles, y en cada una hay tantas rachas de r encargos asignados consecutivamente a ella como el número de elementos de la pila menos $r-1$. Como el tamaño medio de una pila es $\frac{n}{m}$, el número medio de rachas de r encargos en cada pila es $\left(\frac{n}{m}-r+1\right)$. Por último, los r encargos de la racha elegida pueden permutarse de $r!$ formas, por lo que hay $r!$ posibilidades para σ. Así, el entorno r-Permutación en Pila es, en media, de tamaño

$$O\left(m\cdot\left(\frac{n}{m}-r+1\right)\cdot r!\right)=O\big(\,(n-mr+m)\,r!\big)=O(nr!).$$

Por otro lado, Q es una cota superior del tamaño de las pilas, por lo que el tamaño del entorno en el caso peor es $O\big(m\,(Q-r+1)\,r!\big)$. □

4.3.7. r-Permutación Completa en Pila

En el entorno *r-Permutación en Pila*, presentado anteriormente, un movimiento consiste en elegir una racha de r encargos asignados a una misma pila y permutarlos. De esta manera, con un movimiento r-SP se puede optimizar un trozo de una pila, pero no una pila completa. Aumentar mucho el tamaño de r para permitir optimizar trozos más grandes no es conveniente, ya que el tamaño de un entorno r-SP crece exponencialmente con r. Para solucionar este problema surge de forma natural la opción de dividir una pila en varios trozos disjuntos de r encargos y optimizar cada uno por separado, realizando un movimiento r-SP sobre cada uno. De esta manera, el tamaño del entorno sigue siendo manejable y toda la pila entra en juego en la realización del movimiento.

Una pila se puede dividir en rachas disjuntas de r encargos de muchas maneras, siendo posible incluso dividirla en rachas de distintos tamaños y realizar movimientos r-SP con distintos valores de r. La estrategia que se ha considerado más adecuada, por sencillez y utilidad, ha sido la de dividir la pila elegida en tantas *rachas disjuntas consecutivas* como sea posible, no considerándose para la realización del movimiento los últimos encargos asignados a la pila no pertenecientes a ninguna racha, si los hubiera (caso de que el tamaño de la pila no sea divisible por r).

Así, el entorno *r-Permutación Completa en Pila* (*r-Complete Stack Permutation*, r-CSP) de una solución S del DTSPMS, denotado por r-CSP(S), está formado por todas aquellas soluciones que se pueden obtener a partir de S realizando un movimiento r-SP sobre cada una de las rachas disjuntas consecutivas en que pueda dividirse una pila de S empezando por el primer encargo asignado a ella. Un movimiento a través del cual se genera una solución del entorno r-CSP(S) a partir de S se denomina por extensión Movimiento

de r-Permutación Completa en Pila y se denota por r-CSP.

Factibilidad de la solución

Proposición 31. *Sea $S = (\pi_1, \pi_2, \lambda)$ una solución del* DTSPMS *y $k \in P$ una pila de S. La solución S' obtenida a partir de S dividiendo la pila k en varias rachas disjuntas de encargos asignados consecutivamente a ella y realizando un movimiento* r-SP *sobre cada una, es factible y única.*

Demostración

Al no tener ningún encargo común las rachas en las que se divide la pila k elegida, los movimientos r-SP realizados sobre ellas son independientes, obteniéndose la misma solución independientemente del orden en que se realicen. Además, la solución S' siempre es factible, ya que se obtiene a partir de movimientos r-SP, para los cuales está probado que mantienen la factibilidad de la solución. □

Realización de un movimiento

Un movimiento r-CSP queda determinado por una pila y las permutaciones a realizar sobre cada trozo. La definición 31 precisa este tipo de movimientos.

Definición 31. Un *movimiento de r-Permutación Completa en Pila* sobre una solución del DTSPMS, denotado por r-CSP, consiste en elegir una pila y realizar un movimiento r-SP sobre cada una de las rachas disjuntas consecutivas en las que puede dividirse empezando por el primer encargo asignado a ella. Sean $k \in P$ una pila y $\sigma_1, \cdots, \sigma_s$ una colección de s permutaciones del conjunto $A = \{1, \cdots, r\}$. El conjunto $X^{rCSP}_{k,\sigma_1,\cdots,\sigma_s}$ formado por las soluciones sobre las que se puede realizar el movimiento r-CSP que queda determinado por $k, \sigma_1, \cdots, \sigma_s$ se define como:

$$X^{rCSP}_{k,\sigma_1,\cdots,\sigma_s} = \left\{(\pi_1, \pi_2, \lambda) \in X \mid rs \leq |\lambda^{-1}(k)| < r(s+1)\right\}.$$

Sobre este conjunto se define el *operador movimiento*

$$\begin{array}{rccl} r\text{-CSP}[k, \sigma_1, \cdots, \sigma_s] : X^{rCSP}_{k,\sigma_1,\cdots,\sigma_s} & \longrightarrow & X & \\ S & \longmapsto & r\text{-CSP}[k, \sigma_1, \cdots, \sigma_s](S) \equiv \text{sol. tras mov. } r\text{-CSP} & \end{array}$$

que asigna a cada solución S de su dominio la solución que se obtiene realizando sobre S el movimiento de r-Permutación Completa en Pila sobre la pila k en el que se realiza la permutación σ_i sobre la racha i-ésima de la pila k.

Sea $S = (\pi_1, \pi_2, \lambda)$ la solución inicial sobre la que se va a realizar el movimiento, $k \in P$ la pila elegida, $t = |\lambda^{-1}(k)|$ su tamaño, $U = \{u^1, u^2, \cdots, u^t\}$ el conjunto ordenado de encargos asignados a k de manera que para cada $1 \leq j \leq t-1$ se tiene que $\pi_1^{-1}(u^j) < \pi_1^{-1}(u^{j+1})$, $s = \left\lfloor \frac{|\lambda^{-1}(k)|}{r} \right\rfloor$ el número de rachas disjuntas de r encargos en las que se divide la pila k, $h = t - rs$ el número de encargos de la pila k que no entran en el movimiento y $\sigma_1, \cdots, \sigma_s$ las permutaciones a realizar en cada trozo.

En la Tabla 4.1 que se presenta a continuación se detalla cómo se divide la pila k. La racha i-ésima es la r-upla ordenada $(u^{(i-1)r+1}, \cdots, u^{ir})$, cuyo primer encargo se denotará por $v^i = u^{(i-1)r+1}$ y sobre la cual se realizará la permutación σ_i. Los últimos h encargos, $(u^{rs+1}, \cdots, u^t)$, no sufren permutación alguna. En la Figura 4.7 se representa de forma esquemática esta división de la pila k.

nº racha	encargos	1^{er} encargo	permutación
racha 1	$(u^1, \cdots, u^r)$	$v^1 = u^1$	σ_1
racha 2	$(u^{r+1}, \cdots, u^{2r})$	$v^2 = u^{r+1}$	σ_2
$\vdots$	$\vdots$	$\vdots$	$\vdots$
racha s	$(u^{r(s-1)+1}, \cdots, u^{rs})$	$v^s = u^{r(s-1)+1}$	σ_s
resto	$(u^{rs+1}, \cdots, u^t)$	$v^* = u^{rs+1}$	ninguna

Tabla 4.1: Notación en un movimiento r-CSP

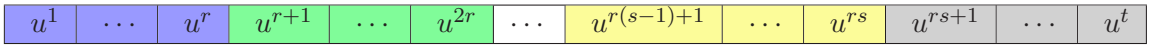


Figura 4.7: División de una pila para la realización de un movimiento r-CSP

Siguiendo esta notación, la proposición 32 detalla cómo construir la solución que se obtiene tras un movimiento r-CSP.

Proposición 32. *La solución* $\hat{S} = (\hat{\pi}_1, \hat{\pi}_2, \hat{\lambda}) = r\text{-CSP}[k, \sigma_1, \cdots, \sigma_s](S)$ *resultado de aplicar el operador movimiento* $r\text{-CSP}[k, \sigma_1, \cdots, \sigma_s]$ *sobre una solución* $S = (\pi_1, \pi_2, \lambda) \in X^{rCSP}_{k,\sigma_1,\cdots,\sigma_s}$ *es la siguiente:*

$$\hat{S} = \left(r\text{-SP}\left[v^s, \sigma_s\right] \circ \cdots \circ r\text{-SP}\left[v^2, \sigma_2\right] \circ r\text{-SP}\left[v^1, \sigma_1\right]\right)(S)$$

Observación 30. Para construir $\hat{S}$ es necesario realizar s movimientos r-SP independientes. Por tanto, al igual que ocurría en este tipo de movimientos, la asignación de pilas queda inalterada, aunque el orden en el que se almacenan los encargos de la pila k sí se modifique.

La Figura 4.8 ilustra cómo realizar un movimiento 3-CSP sobre una solución (π_1, π_2, λ), en concreto sobre una pila con 7 encargos. Dicha pila se divide en tres partes: la primera

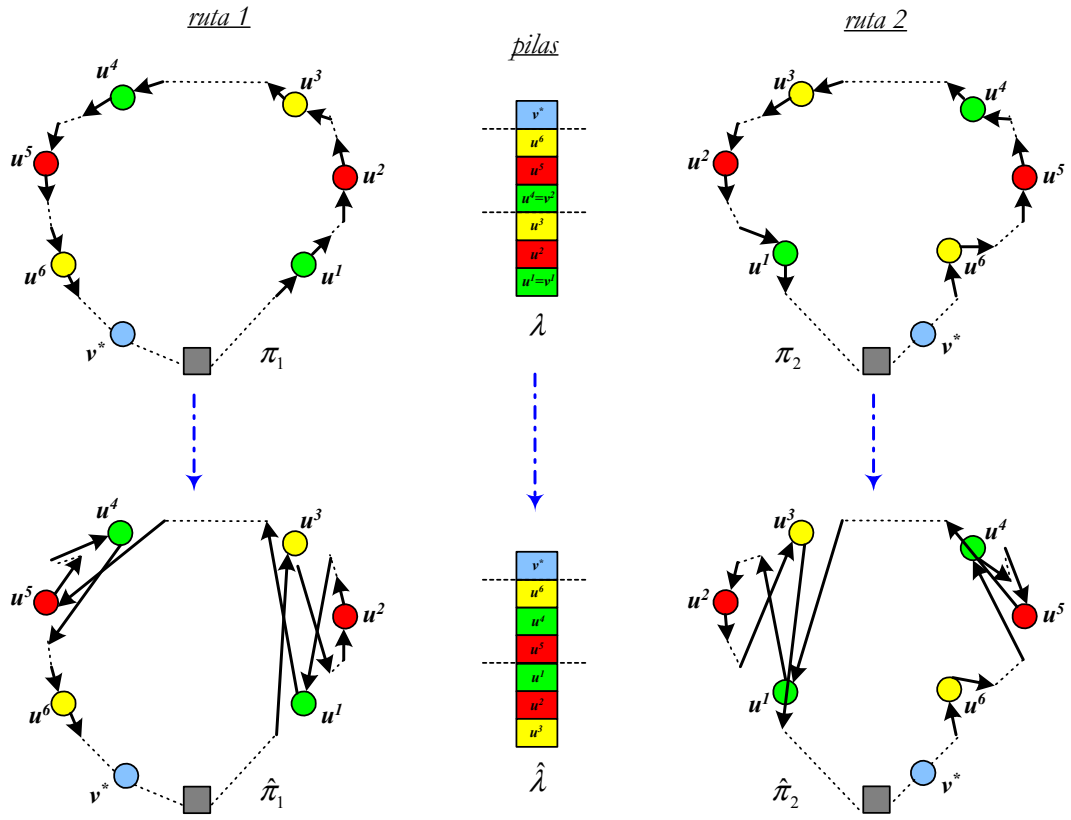


Figura 4.8: Un movimiento de r-Permutación Completa en Pila

la forman los encargos $\{u^1, u^2, u^3\}$, la segunda los encargos $\{u^4, u^5, u^6\}$ y la tercera el encargo $\{u^7\}$. Sobre los dos primeros conjuntos de encargos se realizan sendos movimientos 3-SP independientes, mientras que el encargo u^7 se mantiene en la misma posición. Las permutaciones elegidas para los movimientos 3-SP son $(3, 2, 1)$ y $(2, 1, 3)$, respectivamente, de manera que en la nueva solución obtenida $(\hat{\pi}_1, \hat{\pi}_2, \hat{\lambda})$ el primer grupo de encargos se recoge siguiendo el orden (u^3, u^2, u^1) y el segundo siguiendo el orden (u^5, u^4, u^6).

Representación conjuntista del entorno

El entorno r-CSP de una solución del DTSPMS es el conjunto formado por todas las soluciones que pueden obtenerse a partir de dicha solución realizando un movimiento r-Permutación Completa en Pila. La proposición 33 detalla cómo construir este tipo de entornos.

Proposición 33. *Sean $S = (\pi_1, \pi_2, \lambda) \in X$ una solución factible del DTSPMS , $A = \{1, \cdots, r\}$ y Biy(A) el conjunto de las permutaciones de A. El entorno r-Permutación*

Completa en Pila de la solución S, denotado por r-CSP(S), es el conjunto

$$r\text{-CSP}(S) = \left\{ r\text{-SP}\left[v^s, \sigma_s\right] \circ \cdots \circ r\text{-SP}\left[v^1, \sigma_1\right](S) \,\middle|\, \sigma_1, \cdots, \sigma_s \in Biy(A),\ \exists k \in P \text{ s.a. } C_1, C_2 \right\}$$

donde las condiciones C_i, $i = 1, 2$ son

i. $C_1 : rs \leq |\lambda^{-1}(k)| < r(s+1)$

ii. $C_2 : U = \left\{u^1, u^2, \cdots, u^t\right\}$ *es el conjunto ordenado de encargos asignados a la pila k y $v^1, \cdots, v^s$ se definen según la Tabla 4.1.*

Demostración

Inmediata a partir de la proposición 32. □

Tamaño del entorno

La proposición 34 establece el tamaño de un entorno de r-Permutación Completa en Pila.

Proposición 34. *Sea $S \in X$ una solución factible del DTSPMS. El entorno r-CSP(S) es de tamaño $O(m\,(r!)^{\frac{n}{mr}})$ en media y de tamaño $O(m\,(r!)^{\frac{Q}{r}})$ en el caso peor.*

Demostración

Una vez elegida la pila k sobre la que realizar el movimiento entre las m pilas disponibles, éste queda determinado por las s permutaciones a realizar, siendo t el tamaño de la pila k y $s = \left\lfloor \frac{t}{r} \right\rfloor$, ya que los s trozos en los que se divide la pila k son fijos. Hay $r!$ posibilidades para cada permutación y éstas se pueden elegir de forma independiente, por lo que las s permutaciones pueden elegirse de $(r!)^s$ formas. Además, el tamaño medio de una pila es $O\left(\frac{n}{m}\right)$. Así, el tamaño medio del entorno r-Permutación Completa en Pila es

$$O\left(m \cdot (r!)^s\right) = O\left(m \cdot (r!)^{\frac{t}{r}}\right) = O\left(m \cdot (r!)^{\frac{n}{mr}}\right).$$

Por otro lado, Q es una cota superior de t, por lo que el tamaño del entorno en el caso peor es $O\left(m \cdot (r!)^{\frac{Q}{r}}\right)$. □

4.3.8. Extensión a soluciones α-factibles

Al igual que ocurría con los algoritmos de generación de soluciones iniciales, las estructuras de entornos para soluciones factibles definidas en las secciones anteriores se transforman fácilmente en estructuras de entornos para soluciones α-factibles simplemente considerando una capacidad máxima de $Q + \alpha$ unidades por pila en lugar de Q.

Muy diferente es el caso de las estructuras de entornos para soluciones α-potenciales, que deben definirse específicamente para este espacio de soluciones y que se presentan en la sección siguiente.

4.4. Estructuras de entornos para soluciones α-potenciales

Las estructuras de entornos presentadas en la sección anterior consideran exclusivamente soluciones α-factibles, que garantizan la verificación de las restricciones de precedencia del problema. Sin embargo, en problemas muy restringidos como el DTSPMS resulta útil considerar también soluciones más generales como las α-potenciales, que aunque pueden violar algunas restricciones de precedencia, permiten extender el espacio factible y diversificar el proceso de búsqueda.

En las secciones siguientes se presentan dos nuevas estructuras de entornos sobre el espacio de soluciones α-potenciales del DTSPMS, que se denominarán *Reinserción en Ruta* y *Reinserción en Pila*. Estas estructuras definen dos tipos de movimientos que pueden implementarse fácilmente y aplicarse utilizando poco tiempo de cómputo, permitiendo modificar de forma sencilla tanto las rutas como la asignación de pilas de las soluciones implicadas y cubriendo todo el espacio de soluciones α-potenciales.

4.4.1. Reinserción en Ruta

El entorno *Reinserción en Ruta* (RR: *Route Reinsertion*) de una solución α-potencial S del DTSPMS, denotado por RR(S), está formado por todas aquellas soluciones α-potenciales que se pueden obtener a partir de S eligiendo un encargo y cambiándolo de posición en una de las rutas de S. Al realizar un movimiento de este tipo, denominado por extensión movimiento de Reinserción en Ruta (RR), cambia el coste de la solución y también su infactibilidad.

Un movimiento RR sobre una solución α-potencial $S = (\pi_1, \pi_2, \lambda) \in \tilde{X}_\alpha$ queda definido por el encargo $u \in D$ que se va a recolocar y la nueva posición i^* que va a ocupar en la ruta δ elegida. Al realizar el movimiento sólo cambia π_δ, y por tanto la nueva solución obtenida tras el movimiento seguirá siendo una solución α-potencial para cualquier elección de u, i^*

y δ.

Definición 32. Se define el *operador movimiento*

$$\begin{array}{rcl} \mathrm{RR}[u, i^*, \delta] : \tilde{X}_\alpha & \longrightarrow & \tilde{X}_\alpha \\ S & \longmapsto & \mathrm{RR}[u, i^*, \delta](S) \equiv \text{solución tras movimiento RR} \end{array}$$

que asigna a cada solución α-potencial S la solución que se obtiene realizando sobre S el movimiento RR en el que el encargo u se mueve a la posición i^* de la ruta δ.

La forma precisa de realizar un movimiento RR y construir un entorno de este tipo se detalla en las proposiciones 35 y 36.

Proposición 35. *El resultado de aplicar el operador movimiento* $\mathrm{RR}[u, i^*, \delta]$ *sobre una solución* $S = (\pi_1, \pi_2, \lambda) \in \tilde{X}_\alpha$ *es*

$$RR[u, i^*, \delta] = \begin{cases} (\mathcal{R}^u_{i^*}(\pi_\delta), \pi_2, \lambda) & si \quad \delta = 1 \\ (\pi_1, \mathcal{R}^u_{i^*}(\pi_\delta), \lambda) & si \quad \delta = 2 \end{cases}$$

donde $\mathcal{R}^u_{i^*}$ *es el operador establecido en la definición 28.*

Proposición 36. *El entorno Reinserción en Ruta de una solución* α*-potencial* $S = (\pi_1, \pi_2, \lambda) \in \tilde{X}_\alpha$*, denotado por* $RR(S)$*, es la unión de los entornos* $\mathrm{RR}_1(S)$, $\mathrm{RR}_2(S)$*, definidos como:*

$$\mathrm{RR}_1(S) = \bigcup_{i^* \in I, u \in D} \left\{ (\mathcal{R}^u_{i^*}(\pi_1), \pi_2, \lambda) \right\}, \qquad \mathrm{RR}_2(S) = \bigcup_{i^* \in I, u \in D} \left\{ (\pi_1, \mathcal{R}^u_{i^*}(\pi_2), \lambda) \right\}$$

Así, $\mathrm{RR}(S) = \mathrm{RR}_1(S) \cup \mathrm{RR}_2(S)$.

El tamaño de un entorno de Reinserción en Ruta es cuadrático con respecto al número de encargos del problema, según se muestra en la proposición 37.

Proposición 37. *El entorno* $\mathrm{RR}(S)$ *es de tamaño* $O(n^2)$ *para toda solución* $S \in \tilde{X}_\alpha$.

Demostración

Un movimiento RR queda definido por el encargo $u \in D$ que se va a recolocar, la ruta δ elegida y la nueva posición i^* que va a ocupar en ella, sin ningún otro tipo de restricción adicional. Se pueden elegir n encargos distintos, que pueden reinsertarse en $n-1$ posiciones posibles en cada una de las rutas. Así, el tamaño del entorno será $O(n(n-1)) = O(n^2)$. □

4.4.2. Reinserción en Pila

El entorno *Reinserción en Pila* (SR: *Stack Reinsercion*) de una solución α-potencial S del DTSPMS, denotado por SR(S), está formado por todas aquellas soluciones α-potenciales que se pueden obtener a partir de S eligiendo un encargo y cambiándolo de pila. Al realizar un movimiento de este tipo, denominado por extensión movimiento de Reinserción en Pila (SR), sólo cambia la infactibilidad de la solución, permaneciendo inalterado el coste.

Un movimiento SR sobre una solución α-potencial $S = (\pi_1, \pi_2, \lambda) \in \tilde{X}_\alpha$ queda definido por el encargo $u \in D$ que se va a mover y la pila $k \in P$ en la que se va a recolocar. Al realizar el movimiento sólo cambia λ, y por tanto la nueva solución obtenida tras el movimiento seguirá siendo una solución α-potencial, para cualquier elección de u, siempre y cuando la nueva asignación de pilas no exceda la capacidad máxima $Q + \alpha$. La única pila que crece es k, por lo que es necesario y suficiente comprobar que dicha pila no había alcanzado su capacidad máxima antes del movimiento, esto es $|\lambda^{-1}(k)| < Q + \alpha$.

Definición 33. El conjunto $\tilde{X}^{SR}_{\alpha,u,k}$ formado por las soluciones α-potenciales sobre las que se puede realizar el movimiento SR que queda determinado por el encargo u y la pila k se define como:

$$\tilde{X}^{SR}_{\alpha,u,k} = \left\{(\pi_1, \pi_2, \lambda) \in \tilde{X}_\alpha \text{ tal que } |\lambda^{-1}(k)| < Q + \alpha\right\}.$$

Sobre este conjunto se define el *operador movimiento*

$$\begin{array}{rccl} \mathrm{SR}[u,k] : \tilde{X}^{SR}_{\alpha,u,k} & \longrightarrow & \tilde{X}_\alpha \\ S & \longmapsto & \mathrm{SR}[u,k](S) \equiv \text{solución tras movimiento SR} \end{array}$$

que asigna a cada solución α-potencial S la solución que se obtiene realizando sobre S el movimiento SR en el que el encargo u se mueve a la pila k.

La forma precisa de realizar un movimiento SR y construir un entorno de este tipo se detalla en las proposiciones 38 y 39.

Proposición 38. *El resultado de aplicar el operador movimiento* SR$[u,k]$ *sobre una solución* $S = (\pi_1, \pi_2, \lambda) \in \tilde{X}^{SR}_{\alpha,u,k}$ *es* SR$[u,k](S) = (\pi_1, \pi_2, \hat{\lambda})$*, donde*

$$\hat{\lambda}(w) = \begin{cases} \lambda(w) & si \quad w \in D \backslash \{u\} \\ k & si \quad w = u \end{cases}$$

Proposición 39. *El entorno Reinserción en Pila de una solución* α*-potencial* $S = (\pi_1, \pi_2, \lambda) \in \tilde{X}_\alpha$*, denotado por SR*$(S)$*, es el conjunto*

$$\mathrm{SR}(S) = \left\{(\pi_1, \pi_2, \hat{\lambda}) \in \tilde{X}_\alpha \mid \exists u \in D \text{ tal que } \hat{\lambda}(v) = \lambda(v)\ \forall v \in D \setminus \{u\}\right\}$$

El tamaño de un entorno de Reinserción en Pila es lineal con respecto al producto del número de encargos del problema y el número de pilas disponibles, según muestra la proposición 40.

Proposición 40. *El entorno* SR(S) *es, en el caso peor, de tamaño* $O(nm)$ *para toda solución* $S \in \tilde{X}_\alpha$

Demostración

Un movimiento SR queda definido por el encargo $u \in D$ que se va a mover y la pila $k \in P$ donde se va a recolocar, teniendo en cuenta que la pila k no puede estar llena. Se pueden elegir n encargos distintos, que pueden recolocarse a lo sumo en m pilas distintas. Así, el tamaño del entorno será a lo sumo $O(nm)$. □

Las estrategias de generación de soluciones iniciales y las estructuras de entornos introducidas en este capítulo se utilizarán en el Capítulo 6 para el diseño de algoritmos heurísticos para el DTSPMS. En el próximo capítulo se estudiará cómo tratar la infactibilidad en el ámbito del problema a resolver, con el objetivo de introducir también soluciones α-potenciales en las heurísticas propuestas para flexibilizar el proceso de búsqueda.

Capítulo 5

Tratamiento de la infactibilidad en el DTSPMS

El DTSPMS es un problema con fuertes restricciones de precedencia y capacidad, lo que hace que las soluciones factibles sean difíciles de caracterizar y el proceso de búsqueda local sea complejo. Para pasar de una solución factible a otra mejor es necesario comprobar que el movimiento realizado verifica una serie de condiciones, en muchos casos no triviales, que aseguren que la nueva solución cumple todas las condiciones del problema, lo cual restringe en gran medida el proceso de búsqueda local.

En este tipo de problemas la relajación temporal de algunas restricciones, en nuestro caso las concernientes a la capacidad máxima de las pilas y a las relaciones de precedencia entre encargos asignados a la misma pila, puede ayudar, si se maneja de forma adecuada, a que el proceso de búsqueda sea más sencillo y flexible y se puedan obtener soluciones finales *factibles* de calidad. La relajación de las restricciones de capacidad y precedencias da lugar al uso de soluciones intermedias no factibles, que hemos denominado soluciones α-factibles (definición 20) y α-potenciales (definición 21). En el presente capítulo estudiamos el tratamiento de este tipo de soluciones: cómo medir su infactibilidad, cómo obtener soluciones factibles a partir de soluciones α-factibles y α-potenciales, cómo transformar unas infactibilidades en otras, y en definitiva cómo manejar la introducción de infactibilidad en el proceso de búsqueda para permitir la obtención de buenas soluciones factibles.

En la Sección 5.1 se estudia la infactibilidad producida por la relajación de las restricciones de capacidad, que denominamos *Infactibilidad por Capacidad* (IC). En la Sección 5.2 consideramos el otro tipo de infactibilidad, que se produce por la relajación de las restricciones de precedencia y que denominamos *Infactibilidad por Precedencias* (IP). Por último, en la Sección 5.3 se detalla cómo se relacionan ambos tipos de infactibilidad y cómo pueden combinarse y se propone una nueva función objetivo para guiar el proceso

de búsqueda teniendo en cuenta las medidas de infactibilidad propuestas.

5.1. Infactibilidad por Capacidad: soluciones α-factibles

En la mayoría de las aplicaciones reales del DTSPMS el número de encargos a servir está muy próximo a la capacidad máxima del vehículo, de manera que después de la ruta de recogida el contenedor del vehículo está casi lleno o incluso lleno totalmente. En estas situaciones la capacidad máxima de las pilas está muy ajustada, de manera que en cualquier asignación de pilas factible el espacio libre es escaso o incluso inexistente. Ello provoca que en los movimientos en los que se modifica el tamaño de las pilas (por ejemplo al cambiar de pila un solo encargo) haya poco margen de maniobra, desaprovechándose de alguna manera el potencial de estos movimientos a la hora de diversificar el proceso de búsqueda. Este es el caso, por ejemplo, de los movimientos de Reinserción: si el tamaño de las pilas es ajustado, es decir, $mQ = n$, ningún movimiento de reinserción podrá cambiar la pila del encargo sobre el que se realiza el movimiento, ya que ninguna tendrá espacio libre para acomodarlo; así, la variedad de las soluciones del entorno de Reinserción será mucho menor y el proceso de búsqueda se verá más restringido.

Según nuestra experiencia computacional obtenida resolviendo instancias del DTSPMS con distintas capacidades máximas observamos que las soluciones obtenidas aumentando alguna unidad la capacidad máxima de las pilas son sustancialmente mejores. Además, en algunos casos la solución final obtenida no utiliza en su totalidad la capacidad máxima permitida, o se puede transformar fácilmente y sin coste adicional en otra solución que no la utilice y que por tanto pudiera ser también factible considerando una capacidad máxima más pequeña.

Este hecho sugiere que permitir el uso de alguna unidad adicional en el tamaño máximo de las pilas, introduciendo una infactibilidad en la búsqueda que se denominará *Infactibilidad por Capacidad* (IC), puede mejorar considerablemente el comportamiento de aquellos algoritmos en los que se permiten cambios individuales de pila en el proceso de búsqueda. Las soluciones que utilizan esta capacidad adicional de α unidades por pila se han denominado α-factibles (definición 20). Estas soluciones, que serán no factibles en general, se utilizarán en pasos intermedios en algunas heurísticas, que a pesar de incluir este tipo de soluciones devolverán siempre como resultado final una solución factible (0-factible).

Para manejar adecuadamente la infactibilidad IC es necesario medirla correctamente y diseñar procedimientos con los cuales se pueda reducir el tamaño máximo de las pilas, de manera que se puedan conseguir soluciones factibles a partir de soluciones α-factibles. A continuación se presentan varias medidas para la infactibilidad IC y se detallan diversos procedimientos para eliminarla.

5.1.1. Medidas de la Infactibilidad por Capacidad

Una solución α-factible es infactible si alguna de sus pilas contiene más de Q encargos. En algunas de estas soluciones se puede recuperar la factibilidad realizando modificaciones mínimas, mientras que otras requieren muchos cambios. Al tratar con este tipo de soluciones es importante poder medir de una manera razonable e informativa cuán infactibles son, para así poder evaluar a qué distancia se encuentran de la factibilidad. Para ello se introducen varias *medidas de infactiblidad* que representan de distintas formas el grado de infactibilidad IC de una solución α-factible dada.

Definición 34. Las aplicaciones $\Lambda_C^1, \Lambda_C^2, \Lambda_C^3 : X_\alpha \longrightarrow \mathbb{N} \cup \{0\}$ asignan a cada solución α-factible números enteros no negativos que representan distintas *medidas de su infactibilidad IC*. Para cada $S = (\pi_1, \pi_2, \lambda) \in X_\alpha$, las medidas $\Lambda_C^j(S), j = 1, 2, 3$, se definen de la siguiente forma:

$$\Lambda_C^1(S) = \text{card}\{p \in P | \text{card}(\lambda^{-1}(p)) > Q\}$$

$$\Lambda_C^2(S) = \sum_{p \in P} \text{máx}\{\text{card}(\lambda^{-1}(p)) - Q, 0\}$$

$$\Lambda_C^3(S) = \text{máx}\{0, \max_{p \in P}\{\text{card}(\lambda^{-1}(p))\} - Q\}$$

En la definición 34 se proponen tres medidas $\Lambda_C^1, \Lambda_C^2, \Lambda_C^3$ para la infactibilidad IC, cuya justificación se presenta a continuación.

Según el Modelo 1 (2.11)-(2.20), las únicas restricciones que podría violar una solución α-factible son las restricciones de capacidad, que son de la forma

$$\sum_{i \in D} z_{ip} \leq Q \ \forall p \in P. \tag{5.1}$$

Una solución será más infactible cuanto mayor sea el número de pilas que exceden la capacidad máxima, que coincide con el número de restricciones 5.1 violadas. Esta observación inspira la medida Λ_C^1.

Por otro lado, para eliminar la infactibilidad IC de una solución α-factible habría que recolocar aquellos encargos que excedan la capacidad máxima de las pilas, de manera que cuanto mayor sea el número de éstos mayor será la infactibilidad IC de la solución. La medida Λ_C^2 se basa en esta idea.

Por último, otra forma más sencilla de estimar la infactibilidad IC es considerar simplemente cuántas unidades extra serían necesarias para que la solución dada fuese factible (lo que hemos denominado *margen*), es decir, cuál es el exceso máximo en que incurre la solución, que coincide con el número de encargos que están por encima de la capacidad máxima en la pila más cargada. La medida Λ_C^3 se obtiene a partir de este criterio.

Observación 31. El cálculo de las medidas $\Lambda_C^j(S)$, $j = 1, 2, 3$, es inmediato conociendo el tamaño de las pilas de la solución.

Observación 32. Se tiene que $\Lambda_1(S) \leq m$, $\Lambda_2(S) \leq \alpha m$ y $\Lambda_3(S) \leq \alpha \; \forall S \in X_\alpha$, ya que el tamaño máximo de las pilas de cualquier solución α-factible es a lo sumo $Q + \alpha$.

5.1.2. Algoritmos de Reducción de Pilas

El objetivo de los Algoritmos de Reducción de Pilas que introducimos en esta sección es transformar soluciones α-factibles, con $\alpha \in \mathbb{N}$, en soluciones factibles con el mismo coste. Resulta evidente que no toda solución puede transformarse en otra de idéntico coste y pilas de tamaño máximo más pequeño. Sin embargo, esta operación se puede llevar a cabo en muchos casos. En lo que sigue se presentarán varios algoritmos para llevar a cabo este tipo de operaciones, y nos referiremos al encargo asignado a una determinada pila recogido en último lugar como *encargo cabecero* y a la posición en la que se encuentra en dicha pila como *cabecera*.

5.1.2.1. Reducción en una etapa

Partiendo de una solución S con un tamaño máximo de $Q + \alpha$ encargos por pila, el objetivo de los algoritmos de Reducción en una etapa es encontrar otra solución con las mismas rutas que S (y por tanto el mismo coste) en la que el tamaño máximo de las pilas en la nueva asignación de pilas sea una unidad menor. El esquema general a seguir es el siguiente:

- Primero se identifican las pilas completamente llenas, que son aquellas que tienen asignados $Q + \alpha$ encargos en la solución S y que se denominarán *pilas grandes*. Los encargos cabeceros de las pilas grandes son los elementos que se quieren reasignar, y se van a intentar recolocar en aquellas pilas con un tamaño inferior a $Q + \alpha - 1$, que se denominarán *pilas pequeñas*.
- Una vez identificadas las pilas grandes y pequeñas, para cada pila grande se identifican todas las pilas pequeñas compatibles, que son aquellas que podrían alojar al encargo cabecero de la pila grande de forma factible. La posición en la que quedaría colocado el encargo cabecero de una pila grande en una determinada pila pequeña viene determinada de forma unívoca por las rutas de la solución, siendo necesario comprobar que al recolocarse en dicha posición no se incumplen las restricciones de precedencia y se mantiene la compatibilidad con el resto de encargos asignados a dicha pila.

- Posteriormente, se comprueba si existen asignaciones válidas de pilas grandes a pilas pequeñas compatibles de manera que ninguna pila pequeña exceda la capacidad máxima. Si no existe ninguna asignación de este tipo el algoritmo para: no es posible disminuir el tamaño máximo de las pilas de S.
- Para cada asignación válida, se comprueba si los encargos cabeceros de las pilas grandes asignados a cada pila pequeña verifican dos a dos las relaciones de precedencia. Si ninguna de las asignaciones válidas encontradas respeta las relaciones de precedencia el algoritmo para: no es posible disminuir el tamaño máximo de las pilas de S.
- Cualquier asignación que verifique todas las condiciones anteriores dará lugar al ser implementada a una solución factible con el mismo coste que S y pilas de tamaño máximo una unidad menor.

Sea $\alpha \in \mathbb{N}$ y sea $S \in X_\alpha$ una solución α-factible. El algoritmo **Asignaciones para Reducción en 1 Etapa** (*1-Step Reduction Assignments*, 1-SRA) encuentra todas las formas factibles de recolocar los encargos situados en las cabeceras de aquellas pilas que tienen $Q + \alpha$ encargos respetando las restricciones de precedencia. Al implementar cualquiera de estas recolocaciones se obtienen soluciones $(\alpha - 1)$-factibles con el mismo coste que S. A continuación se presenta la descripción precisa del algoritmo 1-SRA.

Algoritmo 13. Asignaciones para Reducción en 1 Etapa (1-SRA)

Parámetros de entrada

- $S = (\pi_1, \pi_2, \lambda)$: Solución sobre la que realizar la reducción de pilas.
- α: Margen en el tamaño máximo de las pilas con el que se construyó la solución $S \in X_\alpha$.

Parámetros de salida

- A': Conjunto de asignaciones factibles de pilas grandes a pilas pequeñas.

Pseudocódigo

1. *Inicialización:* Poner $q = Q + \alpha$ y $A' = \emptyset$.
2. *Identificación de pilas:*
   - *Pilas grandes:* Poner $P_g = \{k \in P \text{ tales que } |\lambda^{-1}(k)| = q\} \subset P$.
   - *Pilas pequeñas:* Poner $P_p = \{k' \in P \text{ tales que } |\lambda^{-1}(k')| < q - 1\} \subset P$.
3. *Determinación de cabeceras:* Determinar el encargo c_k situado en la cabecera de cada pila grande $k \in P_g$. Poner $C = \{c_k | \, k \in P_g\}$.

4. *Posibles destinos de los encargos cabeceros:* Para cada $k \in P_g$ construir el conjunto $Dest(k) \subset P_p$ formado por todas las pilas pequeñas a las que se puede reasignar el encargo cabecero c_k de la pila k de forma factible.

5. *Comprobación de existencia:* Si existe $k \in P_g$ tal que $Dest(k) = \emptyset$, FIN. El encargo cabecero de la pila k no puede reasignarse a ninguna pila pequeña.

6. *Asignaciones de encargos cabeceros a pilas pequeñas:* Construir el conjunto

$$A = \left\{\xi : C \longrightarrow P_p \mid \xi(c_k) \in Dest(k)\ \forall c_k \in C,\ |\xi^{-1}(k')| + |\lambda^{-1}(k')| \leq q - 1\ \forall k' \in P_p\right\}.$$

7. *Comprobación de existencia:* Si $A = \emptyset$, FIN. No hay ninguna asignación factible de las pilas grandes a las pilas pequeñas.

8. *Relaciones de precedencia:* Construir el conjunto $A' \subset A$ formado por aquellas $\xi \in A$ tales que, para cada $k' \in P_p$, los encargos del conjunto $\xi^{-1}(k')$ verifican dos a dos las relaciones de precedencia impuestas por las rutas π_1 y π_2 al ser reasignadas a la pila k'.

El algoritmo 1-SRA devuelve un conjunto A' con todas las formas posibles de recolocar los encargos cabeceros de la solución dada para disminuir en una unidad el tamaño máximo de las pilas. El algoritmo de **Reducción en 1 Etapa** (*1-Step Reduction*, 1-SR) que se detalla a continuación toma como parámetro un subconjunto de A' con algunas recolocaciones factibles e implementa una de ellas, devolviendo una solución con el mismo coste que la inicial y pilas de tamaño máximo una unidad menor.

Algoritmo 14. Reducción en 1 Etapa (1-SR)

Parámetros de entrada

- $S = (\pi_1, \pi_2, \lambda)$: Solución sobre la que realizar la reducción de pilas.
- α: Margen en el tamaño máximo de las pilas con el que se construyó la solución $S \in X_\alpha$.
- A': Conjunto de asignaciones factibles de pilas grandes a pilas pequeñas que pueden ser utilizadas por el algoritmo.

Parámetros de salida

- *haySol*: Variable booleana con valor T si la búsqueda de la solución S^* ha concluido con éxito y F en caso contrario.
- $S^* = (\pi_1, \pi_2, \lambda^*) \in X_{\alpha-1}$: Solución con el mismo coste que S y tamaño máximo por pila $Q + \alpha - 1$, si se encuentra.
- A'': Conjunto de asignaciones factibles de pilas grandes a pilas pequeñas no utilizadas.

Pseudocódigo

1. *Comprobación de asignaciones factibles:* Si $A' = \emptyset$ poner $haySol = F$, $A'' = \emptyset$, FIN. No hay ninguna asignación factible.
2. *Implementación de la reducción de las pilas:*
   2.1. Poner $haySol = T$.
   2.2. Elegir aleatoriamente una asignación factible $\xi \in A'$.
   2.3. Determinar el encargo c_k situado en la cabecera de cada pila grande $k \in P$ tal que $|\lambda^{-1}(k)| = Q + \alpha$. Poner $C = \{c_k : |\lambda^{-1}(k)| = Q + \alpha\}$.
   2.4. Construir la asignación de pilas λ^* de la nueva solución $S^* = (\pi_1, \pi_2, \lambda^*)$ que queda determinada por ξ haciendo:

$$\lambda^*(c_k) = \xi(c_k) \;\; \forall c_k \in C\,, \qquad \lambda^*(u) = \lambda(u) \;\; \forall u \in D \backslash C$$

   2.5. Poner $A'' = A' \backslash \{\xi\}$.

Los algoritmos 1-SRA y 1-SR deben utilizarse conjuntamente para encontrar las asignaciones factibles y realizar los cambios correspondientes en la solución dada. El algoritmo de **Reducción Completa en 1 Etapa** (*1-Step Complete Reduction*, 1-SCR) que se presenta a continuación combina ambos algoritmos y funciona de forma autónoma.

Algoritmo 15. Reducción Completa en 1 Etapa (1-SCR)

Parámetros de entrada

- $S = (\pi_1, \pi_2, \lambda)$: Solución sobre la que realizar la reducción de pilas.
- α: Margen en el tamaño máximo de las pilas con el que se construyó la solución $S \in X_\alpha$.

Parámetros de salida

- $haySol$: Variable booleana con valor T si la búsqueda de la solución S^* ha concluido con éxito y F en caso contrario.
- $S^* = (\pi_1, \pi_2, \lambda^*) \in X_{\alpha-1}$: Solución con el mismo coste que S y tamaño máximo por pila $Q + \alpha - 1$, si se encuentra.

Otros algoritmos utilizados

- 1-SRA, 1-SR.

Pseudocódigo

1. *Búsqueda de asignaciones:* A=1-SRA(S,α).
2. *Implementación de cambios:* ($haySol$,S^*,A)=1-SR(S,α,A).

5.1.2.2. Reducción multietapa

El algoritmo 1-SCR reduce en una unidad, si es posible, el tamaño máximo de las pilas de la solución, consiguiendo transformar soluciones α-factibles en $(\alpha-1)$-factibles. Sin embargo, si se aplica varias veces de forma consecutiva el tamaño máximo de las pilas podría reducirse en varias unidades, obteniéndose soluciones α^*-factibles con $0 \leq \alpha^* < \alpha$.

El algoritmo de **Reducción Multi-Etapa** (*Multi-Step Reduction*, MSR) que se presenta a continuación sigue esta idea: si en cada iteración se consigue reducir en una unidad el tamaño máximo de las pilas de la solución, se aplica de nuevo el algoritmo 1-SCR hasta que se alcanza el tamaño máximo deseado. El objetivo es conseguir una solución $S^* \in X_{\alpha^*}$ con $\alpha^* < \alpha$, y en caso de que en algún nivel la reducción no pueda realizarse el algoritmo para y devuelve una solución $\hat{\alpha}$-factible, con $\hat{\alpha} \geq \alpha^*$.

Algoritmo 16. Reducción Multi-Etapa (MSR)

Parámetros de entrada

- $S = (\pi_1, \pi_2, \lambda)$: Solución sobre la que realizar la reducción de pilas.
- α: Margen en el tamaño máximo de las pilas con el que se construyó la solución $S \in X_\alpha$.
- α^*: Margen en el tamaño máximo de las pilas que se desea conseguir.

Parámetros de salida

- $\hat{\alpha}$: Margen en el tamaño máximo de las pilas de la solución S^* construida por el algoritmo. Se verifica que $\alpha^* \leq \hat{\alpha} \leq \alpha$.
- S^*: Solución con el mismo coste que S y tamaño máximo por pila $Q+\hat{\alpha}$. Se consigue la reducción deseada si $\hat{\alpha} = \alpha^*$.

Otros algoritmos utilizados

- 1-SCR.

Pseudocódigo

1. *Inicialización:* Poner $q = Q + \alpha$, $q^* = Q + \alpha^*$ y $\varepsilon = \alpha - \alpha^*$.
2. *Reducir una unidad:* $(haySol, \hat{S})$ =1-SCR$(S, \alpha^* + \varepsilon)$.
3. *Comprobación de la reducción:* Si $haySol = F$, FIN. Poner $S^* = S$ y $\hat{\alpha} = \alpha^* + \varepsilon$. No se ha conseguido una solución factible. La mejor solución conseguida es S^*, con tamaño máximo de pilas $Q + \hat{\alpha}$.
4. *Actualización:* Poner $S = \hat{S}$, $\varepsilon = \varepsilon - 1$.

5. *Iteración:* Si $\varepsilon > 0$ volver al paso 2. En otro caso: FIN. Poner $S^* = S$ y $\hat{\alpha} = \alpha^*$. La solución S^* es una solución factible con el mismo coste que la inicial y tamaño máximo por pila $Q + \alpha^*$.

5.1.2.3. Reducción multietapa con paso atrás

En cada iteración del algoritmo MSR en la que se produce una reducción del tamaño máximo de las pilas, el algoritmo 1-SCR elige al azar una de las configuraciones posibles a partir de las cuales se puede obtener dicha reducción y la implementa, tomándose la solución obtenida como punto de partida para la siguiente iteración. Es posible que en las iteraciones posteriores no exista ninguna configuración factible que permita disminuir de nuevo el tamaño de las pilas, pero podría ser que de haber elegido una configuración diferente en iteraciones anteriores, el proceso iterativo de reducción hubiera podido finalizar con éxito. Esto se debe a que el algoritmo MSR no examina todas las configuraciones posibles a partir de las cuales se puede reducir un determinado número de unidades el tamaño máximo de las pilas, por lo que es posible que exista una reducción factible pero el algoritmo no la encuentre.

Para cubrir todas las configuraciones posibles habría que reiniciar el proceso de reducción iterativa cada vez que en un determinado nivel no exista ninguna configuración válida para disminuir en una unidad el tamaño máximo actual, de manera que se reconsideraran en cada nivel las configuraciones factibles que no han sido utilizadas previamente hasta que se llegara a una solución factible o se agotaran todas las configuraciones factibles en todos los niveles. El algoritmo que se presenta a continuación, llamado **Reducción Multi-Etapa con Paso Atrás** (*Multi-Step Reduction with Backtracking*, MSRB), sigue este proceso para encontrar, si existe, una solución $S^* \in X_{\alpha^*}$ con $\alpha^* < \alpha$.

Algoritmo 17. Reducción Multi-Etapa con Paso Atrás (MSRB)

Parámetros de entrada

- $S = (\pi_1, \pi_2, \lambda)$: Solución sobre la que realizar la reducción de pilas.
- α: Margen en el tamaño máximo de las pilas con el que se construyó la solución $S \in X_\alpha$.
- α^*: Margen en el tamaño máximo de las pilas que se desea conseguir.

Parámetros de salida

- $\hat{\alpha}$: Margen en el tamaño máximo de las pilas de la solución S^* construida por el algoritmo. Se verifica que $\alpha^* \leq \hat{\alpha} \leq \alpha$.

- S^*: Solución con el mismo coste que S y tamaño máximo por pila $Q+\hat{\alpha}$. Se consigue la reducción deseada si $\hat{\alpha} = \alpha^*$.

Otros algoritmos utilizados

- 1-SRA, 1-SR.

Pseudocódigo

1. *Inicialización:* Poner $q = Q + \alpha$, $q^* = Q + \alpha^*$, $\varepsilon = \alpha - \alpha^*$, $\hat{\alpha} = \alpha$ y $S^{\hat{\alpha}} = S$.
2. *Reducir una unidad:* Poner $S_\varepsilon = S$ y hacer $(haySol, \hat{S}, A'_\varepsilon)$ =1-SR($S, \alpha^* + \varepsilon$,1-SRA($S, \alpha^* + \varepsilon$)).
3. *Volver atrás:* Si $haySol = F$,
   - Si $\varepsilon = \alpha - \alpha^*$: FIN. Se han examinado todas las posibilidades y no se ha conseguido una solución factible. La mejor solución conseguida es $S^* = S^{\hat{\alpha}} \in X_{\hat{\alpha}}$.
   - Si $A'_{\varepsilon+1} = \emptyset$, poner $\varepsilon = \varepsilon + 1$ y volver al paso 3.
   - Si $A'_{\varepsilon+1} \neq \emptyset$:
     - Poner $\varepsilon = \varepsilon + 1$.
     - Hacer $(haySol, \hat{S}, A'_\varepsilon)$ =1-SR($S_\varepsilon, \alpha^* + \varepsilon, A'_\varepsilon$).
4. *Actualización:* Poner $S = \hat{S}$, $\varepsilon = \varepsilon - 1$.
5. *Solución más reducida:* Si $\hat{\alpha} > \alpha^* + \varepsilon$, poner $\hat{\alpha} = \alpha^* + \varepsilon$, $S^{\hat{\alpha}} = S$.
6. *Reducir más:*
   - Si $\varepsilon > 0$ volver al paso 2.
   - En otro caso: FIN. Poner $S^* = S$ y $\hat{\alpha} = \alpha^*$. La solución S^* es una solución factible con el mismo coste que la inicial.

5.1.2.4. Reducción global con tamaño fijo

El algoritmo MSRB presentado anteriormente es computacionalmente muy complejo, ya que maneja múltiples configuraciones en cada nivel y tiene que avanzar y retroceder de uno a otro constantemente, considerando una cantidad de configuraciones que crece exponencialmente con el número de niveles. Es interesante desde un punto de vista teórico, ya que generaliza al algoritmo MSR, pero su implementación práctica es extremadamente complicada.

Resulta más adecuado y eficiente generalizar el algoritmo de reducción en 1 etapa de otra forma, enfocando el proceso de reducción de forma global de manera que se pueda

reducir el tamaño máximo de las pilas un determinado número de unidades de una sola vez. Los parámetros de entrada y salida de los algoritmos MSR y MSRB se mantienen, pero ahora la reorganización de los encargos que ocupan posiciones que exceden el tamaño máximo prefijado se realiza en un solo paso. El algoritmo denominado **Reducción Global con Tamaño Fijo** (*Global Reduction with Fixed Size*, GRFS) que se presenta a continuación se basa en esta idea y devuelve, si existe, una solución $S^* \in X_{\alpha^*}$ con $\alpha^* < \alpha$. Realizando un pequeño abuso del lenguaje, en los algoritmos de reducción global llamaremos *encargos cabeceros* a los encargos situados en las *últimas* posiciones de una pila, por encima del nivel considerado en cada momento, y no sólo a los situados en la *última* posición como se había hecho hasta ahora.

Algoritmo 18. Reducción Global con Tamaño Fijo (GRFS)

Parámetros de entrada

- $S = (\pi_1, \pi_2, \lambda)$: Solución sobre la que realizar la reducción de pilas.
- α: Margen en el tamaño máximo de las pilas con el que se construyó la solución $S \in X_\alpha$.
- α^*: Margen en el tamaño máximo de las pilas que se desea conseguir.

Parámetros de salida

- $\hat{\alpha}$: Margen en el tamaño máximo de las pilas de la solución S^* construida por el algoritmo. Se verifica que $\alpha^* \leq \hat{\alpha} \leq \alpha$.
- S^*: Solución con el mismo coste que S y tamaño máximo por pila $Q + \hat{\alpha}$. Se consigue la reducción deseada si $\hat{\alpha} = \alpha^*$.

Pseudocódigo

1. *Inicialización:* Poner $q^* = Q + \alpha^*$, $\hat{\alpha} = \alpha$ y $S^* = S$.
2. *Identificación de pilas:*
   - *Pilas grandes:* Poner $P_g = \left\{k \in P \text{ tales que } t_k = |\lambda^{-1}(k)| > q^*\right\} \subset P$.
   - *Pilas pequeñas:* Poner $P_p = \left\{k' \in P \text{ tales que } |\lambda^{-1}(k')| < q^*\right\} \subset P$.
3. *Determinación de elementos cabeceros:* Para cada pila grande $k \in P_g$ determinar los $\varepsilon_k = t_k - q^*$ encargos $\left\{c_1^k, \cdots, c_{\varepsilon_k}^k\right\} = C^k$ que ocupan posiciones por encima de q^* en la pila k. Construir el conjunto $C = \bigcup_{k \in P_g} C^k$, formado por los encargos que deben ser recolocados en las pilas pequeñas.
4. *Posibles destinos de los elementos cabeceros:* Para cada $c \in C$ construir el conjunto $Dest(c) \subset P_p$ formado por todas las pilas pequeñas a las que se puede reasignar el encargo c de forma factible.

5. *Comprobación de existencia:* Si existe un encargo cabecero $c \in C$ tal que $Dest(c) = \emptyset$, FIN. El encargo c no puede reasignarse en ninguna pila pequeña.

6. *Asignaciones de elementos cabeceros a pilas pequeñas:* Construir el conjunto

$$A = \left\{\xi : C \longrightarrow P_p \mid \xi(c) \in Dest(c)\ \forall c \in C,\ |\xi^{-1}(k')| + |\lambda^{-1}(k')| \leq q^*\ \forall k' \in P_p\right\}.$$

7. *Comprobación de existencia:* Si $A = \emptyset$, FIN. No hay ninguna asignación factible de elementos cabeceros a las pilas pequeñas.

8. *Relaciones de precedencia:* Construir el conjunto $A' \subset A$ formado por aquellas $\xi \in A$ tales que, para cada pila pequeña $k' \in P_p$, los encargos cabeceros del conjunto $\xi^{-1}(k')$ verifican dos a dos las relaciones de precedencia impuestas por las rutas π_1 y π_2 al ser reasignados a la pila k'.

9. *Comprobación de existencia:* Si $A' = \emptyset$, FIN. No hay ninguna asignación factible de los elementos cabeceros a las pilas pequeñas que respete las relaciones de precedencia.

10. *Implementación de la reducción de las pilas:*

    10.1. Poner $\hat{\alpha} = \alpha^*$.

    10.2. Elegir una asignación factible $\xi \in A'$.

    10.3. Construir la asignación de pilas λ^* de la nueva solución $S^* = (\pi_1, \pi_2, \lambda^*)$ que queda determinada por ξ haciendo:

$$\lambda^*(c) = \xi(c)\ \ \forall c \in C\,, \qquad \lambda^*(u) = \lambda(u)\ \ \forall u \in D \backslash C$$

5.1.2.5. Reducción global con tamaño ajustable

Cuando el algoritmo de reducción global GRFS no encuentra ninguna recolocación factible para conseguir la reducción requerida, simplemente devuelve la solución inicial sin realizar cambio alguno. Por otro lado, el algoritmo MSR, al ir reduciendo el tamaño máximo de las pilas una unidad en cada paso, cuando no consigue la reducción total exigida puede ser capaz de disminuir dicho tamaño en *algunas* unidades, devolviendo una solución que, aunque todavía es infactible, tiene una medida de infactibilidad menor, pudiendo ser convertida en factible posteriormente con mayor facilidad. Esta reducción paulatina de la infactibilidad proporciona mayor eficacia al proceso de búsqueda, por lo que sería interesante incluirla en el algoritmo de reducción global.

El algoritmo de **Reducción Global con Tamaño Ajustable** (*Global Reduction with Adjustable Size*, GRAS) que se presenta a continuación llama al algoritmo de reducción global GRFS para buscar una recolocación factible con la que conseguir el tamaño máximo exigido α^*, pero si no se encuentra tal recolocación, vuelve a llamar al algoritmo GRFS con

valores de α^* cada vez mayores para ir disminuyendo la exigencia y, si es posible, reducir el tamaño máximo de las pilas en alguna unidad.

Algoritmo 19. Reducción Global con Tamaño Ajustable (GRAS)

Parámetros de entrada

- $S = (\pi_1, \pi_2, \lambda)$: Solución sobre la que realizar la reducción de pilas.
- α: Margen en el tamaño máximo de las pilas con el que se construyó la solución $S \in X_\alpha$.
- α^*: Margen en el tamaño máximo de las pilas que se desea conseguir.

Parámetros de salida

- $\hat{\alpha}$: Margen en el tamaño máximo de las pilas de la solución S^* construida por el algoritmo. Se verifica que $\alpha^* \leq \hat{\alpha} \leq \alpha$.
- S^*: Solución con el mismo coste que S y tamaño máximo por pila $Q + \hat{\alpha}$. Se consigue la reducción deseada si $\hat{\alpha} = \alpha^*$.

Otros algoritmos utilizados

- GRFS

Pseudocódigo

1. *Búsqueda de solución factible:* Hacer $(\bar{\alpha}, S^*)$ =GRFS(S, α, α^*).
   - Poner $\hat{\alpha} = \alpha^*$.
   - Si $\bar{\alpha} = \alpha^*$, FIN. Se devuelve la solución $S^* \in X_{\alpha^*}$.
2. *Relajación del problema:*
   - Si $\hat{\alpha} < \alpha$, poner $\hat{\alpha} = \hat{\alpha} + 1$; en otro caso, FIN: no se ha podido reducir el tamaño máximo de las pilas de la solución S en ninguna unidad: $S^* = S$ y $\hat{\alpha} = \alpha$.
3. *Búsqueda de solución con margen $\hat{\alpha}$:* Hacer $(\bar{\alpha}, \hat{S})$ =GRFS$(S, \alpha, \hat{\alpha})$.
4. *Comprobación de la reducción:*
   - Si $\bar{\alpha} = \hat{\alpha}$, FIN. Poner $S^* = \hat{S}$. No se ha conseguido la reducción máxima, la mejor solución conseguida es $S^* \in X_{\hat{\alpha}}$.
   - En otro caso, volver al paso 2.

5.1.3. Complejidad de los Algoritmos de Reducción de Pilas

En las proposiciones que se presentan a continuación se analiza la complejidad de los distintos algoritmos de Reducción de Pilas propuestos en la sección anterior.

Proposición 41. *Sea $S \in X_\alpha$ una solución α-factible de un DTSPMS con m pilas, $\alpha^* < \alpha$ y $\varepsilon = \alpha - \alpha^*$. El coste del algoritmo* GRFS(S, α, α^*) *es, en el caso peor,* $O\left(\varepsilon \log(\varepsilon m)\, m^{\varepsilon m+1}\right)$.

Demostración

El objetivo del algoritmo GRFS(S, α, α^*) es transformar la solución $S \in X_\alpha$ en α^*-factible, para lo cual habrá que recolocar un máximo de ε encargos por pila. Así, el número de encargos a recolocar, que se denominarán *encargos cabeceros*, está acotado por εm. Cada uno de ellos puede reasignarse a cualquiera de las $m-1$ pilas restantes con las que sea compatible, por lo que hay, a lo sumo, $(m-1)^{\varepsilon m}$ formas posibles de asignar los εm encargos cabeceros a pilas compatibles en las que se puedan realojar. Dada una de estas $(m-1)^{\varepsilon m}$ configuraciones, es necesario comprobar que los clientes cabeceros asignados a cada pila son compatibles con las rutas de la solución, lo cual ocurre si los órdenes relativos de dichos encargos en ambas rutas son inversos. Para hacer esta comprobación lo más eficiente es ordenar los εm encargos según sus posiciones en la ruta 1 (lo cual requiere $O(\varepsilon m \log(\varepsilon m))$ operaciones) y posteriormente ver si los órdenes de los encargos reasignados a cada pila son inversos a los obtenidos en la ruta 2 (número de operaciones proporcional a εm), lo cual da un coste total $O(\varepsilon m \log(\varepsilon m)) + O(\varepsilon m) = O(\varepsilon m \log(\varepsilon m))$.

Así, el número de operaciones elementales necesarias para examinar todas las configuraciones posibles, $O((m-1)^{\varepsilon m})$, e identificar cuáles son factibles, $O(\varepsilon m \log(\varepsilon m))$, es, en el caso peor, del orden del producto de estas dos cantidades:

$$O((m-1)^{\varepsilon m} \cdot \varepsilon m \log(\varepsilon m)) = O(\varepsilon m \log(\varepsilon m)\, m^{\varepsilon m}) = O(\varepsilon \log(\varepsilon m)\, m^{\varepsilon m+1}).$$

Determinando la complejidad de cada paso en el pseudocódigo del algoritmo 18 se llega a la misma conclusión. Los pasos 1, 3, 5, 6 y 9 tienen un coste $O(1)$, el paso 2 es $O(m)$, el paso 4 es $O(m^2\alpha)$, el paso 6 es $O(m^{\varepsilon m})$, el paso 8 es $O\left(\varepsilon \log(\varepsilon m)\, m^{\varepsilon m+1}\right)$ y el paso 10 es $O(\varepsilon m)$. El paso 8 tiene un coste de orden superior al de todos los demás, y por tanto el coste total del algoritmo viene determinado por él. □

Observación 33. Según la proposición 41, el coste del algoritmo GRFS no depende del número total de encargos n, sino que sólo depende del número de pilas m y del número de unidades ε que se quiere reducir el tamaño máximo de las pilas. El algoritmo es exponencial en ε y m, pero ambas cantidades suelen ser pequeñas y pueden mantenerse fijas aunque aumente el tamaño del problema.

Proposición 42. *Sea $S \in X_\alpha$ una solución α-factible de un DTSPMS con m pilas, $\alpha^* < \alpha$ y $\varepsilon = \alpha - \alpha^*$. Los costes de los algoritmos de reducción de pilas en el caso peor son:*

Algoritmo	*Coste*
1-SCR	$O(m^{m+1}\log m)$
MSR	$O(\varepsilon m^{m+1}\log m)$
MSRB	$O(m^{\varepsilon m+1}\log m)$
GRFS	$O\left(\varepsilon m^{\varepsilon m+1}\log \varepsilon m\right)$
GRAS	$O\left(\varepsilon m^{\varepsilon m+1}\log \varepsilon m\right)$

Demostración

Para obtener todos estos resultados se utiliza la proposición 41, que establece que el coste del algoritmo GRFS es $O\left(\varepsilon m^{\varepsilon m+1}\log \varepsilon m\right)$.

- El coste del algoritmo 1-SCR es el mismo que el de GRFS con $\varepsilon = 1$, y por tanto su coste es $O(m^{m+1}\log m)$ en el caso peor.
- El algoritmo MSR llama al algoritmo 1-SR a lo sumo ε veces; por tanto, su coste es $O(\varepsilon m^{m+1}\log m)$ en el caso peor.
- El algoritmo MSRB examina m^m configuraciones en cada nivel, cada una de las cuales tiene un coste asociado de chequeo de factibilidad que es $O(m\log m)$, y considera todas las formas posibles de elegir una configuración en cada uno de estos ε niveles. Así, se tienen que examinar $(m^m)^\varepsilon = m^{\varepsilon m}$ combinaciones con un coste $O(m\log m)$, lo que da un coste final de $O(m^{\varepsilon m+1}\log m)$ en el caso peor.
- El coste del algoritmo GRAS es la suma de los costes del algoritmo GRFS para los distintos valores de $\alpha^* \in \{\alpha^*, \alpha^*+1, \cdots, \alpha-1\}$, que son los costes de intentar reducir el tamaño máximo de las pilas en $\varepsilon, \varepsilon-1, \cdots, 1$ unidades. Así, dicho coste es

$$O\Bigg(\underbrace{\log m \cdot m^{m+1} + \cdots + (\varepsilon-1)\log\left((\varepsilon-1)m\right)\cdot m^{(\varepsilon-1)m+1}}_{\varphi} + \underbrace{\varepsilon \log \varepsilon m \cdot m^{\varepsilon m+1}}_{\mu}\Bigg)$$

Sacando factor común una potencia de m y dado que

$$z\log zm < \varepsilon \log \varepsilon m,\ \forall z = 1, 2, \cdots \varepsilon - 1 \tag{5.2}$$

se tiene que φ verifica

$$\varphi < \varepsilon m \log \varepsilon m \cdot \left(m^m + \cdots + m^{(\varepsilon-1)m}\right) \tag{5.3}$$

Sustituyendo $x = m^m$ en la identidad

$$x^{\varepsilon} - 1 = (1 + x + \cdots + x^{\varepsilon-1})(x - 1) \tag{5.4}$$

se tiene que

$$\begin{aligned} m^{\varepsilon m} - 1 = (1 + m^m + \cdots + m^{(\varepsilon-1)m})(m^m - 1), \text{ luego } \mu = \varepsilon m \log \varepsilon m \cdot m^{\varepsilon m} = \\ = \varepsilon m \log \varepsilon m \cdot \left(\left(1 + m^m + \cdots + m^{(\varepsilon-1)m}\right)(m^m - 1) + 1 \right) = \\ = \varepsilon m \log \varepsilon m \cdot \left(\left(m^m + \cdots + m^{(\varepsilon-1)m}\right)(m^m - 1) + m^m - 1 + 1 \right) = \\ = \varepsilon m \log \varepsilon m \cdot m^m + \underbrace{\varepsilon m \log \varepsilon m \cdot \left(m^m + \cdots + m^{(\varepsilon-1)m}\right)}_{> \varphi \text{ por (5.3)}} (m^m - 1) > \end{aligned}$$

$$> \varepsilon \log \varepsilon m \cdot m^{m+1} + \varphi\,(m^m - 1) \gg \varphi \quad \Rightarrow \quad \mu \gg \varphi \tag{5.5}$$

De (5.5) se deduce que el coste de intentar reducir el tamaño de las pilas de la solución dada en $1, 2, \cdots, \varepsilon - 1$ unidades utilizando el algoritmo GRFS es sensiblemente menor que el de intentar reducir dicho tamaño en ε unidades utilizando el mismo algoritmo. Así, $O(\varphi + \mu) = O(\mu) = O\left(\varepsilon m^{\varepsilon m+1} \log \varepsilon m\right)$, de manera que, en el caso peor, el coste de los algoritmos GRAS y GRFS es el mismo.

Con lo que quedan demostrados todos los resultados propuestos. □

Observación 34. Como era obvio, el algoritmo 1-SCR es el de menor complejidad, seguido de MSR, que aplica 1-SR ε veces. Ambos algoritmos tienen un coste exponencial en m, pero no en ε. Los otros tres algoritmos, sin embargo, presentan un coste que crece exponencialmente respecto a ambas variables, lo cual hace que tengan una complejidad mayor.

Observación 35. Según la proposición 42, el coste del algoritmo MSRB es teóricamente del mismo orden que el de los algoritmos GRFS y GRAS, e incluso algo menor debido al factor ε que aparece multiplicando y dentro del logaritmo en las expresiones del coste de los dos últimos. Sin embargo estos costes están calculados para el *caso peor*, y en la práctica se observa que el tiempo de cómputo necesario para resolver un problema con los algoritmos GRFS y GRAS es en realidad mucho menor que utilizando MSRB.

Conclusión: Los algoritmos que utilizaremos habitualmente en la práctica para reducir eficientemente el tamaño de las pilas de una solución lo suficiente como para hacerla factible, y si esto no se puede, reducir tantas unidades como sea posible, serán los algoritmos MSR y GRAS. El tiempo de cómputo necesario para el primero es menor que para el segundo, ya que éste realiza una reorganización global de los α niveles.

5.1.4. Relación entre los Algoritmos de Reducción de Pilas

Los algoritmos de reducción de pilas introducidos anteriormente van encaminados a reducir el tamaño máximo de las pilas de una solución, aunque esta reducción no tiene por qué ser completa. El algoritmo 1-SCR busca reducir el tamaño máximo de las pilas en una sola unidad, mientras que el algoritmo GRFS busca la reducción de un número de unidades fijo, no realizando ninguna modificación en la solución inicial si no consigue dicha reducción. En una categoría aparte se encuentran los algoritmos MSR, MSRB y GRAS, que intentan conseguir una reducción de cuantas unidades sea posible en caso de no lograr la reducción inicialmente deseada. En lo que sigue nos referiremos a estos tres últimos algoritmos como *Algoritmos de Reducción de Tamaño Variable* (RTV), y estudiaremos la relación que hay entre ellos.

Los algoritmos GRAS y MSRB son determinísticos en el sentido de que, dadas una instancia del DTSPMS y una solución S sobre la que se quiere reducir el tamaño máximo de las pilas en α unidades, las reorganizaciones de las pilas de S que exploran son siempre las mismas para cualquier ejecución, aunque finalmente sólo se implemente una de ellas. Por el contrario, el algoritmo MSR elige aleatoriamente, de entre todas las factibles, una configuración en cada uno de los α niveles, de manera que las reorganizaciones consideradas en un nivel dependen de las realizadas en niveles anteriores. Así, para fijar las configuraciones consideradas por el algoritmo MSR en una ejecución concreta es necesario definir las *elecciones* que se van a realizar en cada uno de los α niveles. Si, como es habitual, las reorganizaciones de cada nivel se recorren ordenadamente, bastaría con elegir el índice que ocupa la reorganización elegida en cada nivel.

Definición 35. Sea $S \in X_\alpha$ una solución α-factible del DTSPMS. Se dice que el vector $(t_1, \cdots, t_\alpha) \in \mathbb{N}^\alpha$ es un *trazado* del algoritmo MSR de tamaño α para la solución S si:

i) $t_1 > 0$ y es menor o igual que el número de reordenaciones factibles en el primer nivel (ó $t_1 = 0$ si no existen reordenaciones factibles en dicho nivel).

ii) Para cada $j = 2, \cdots, \alpha$, tras la reordenación t_i-ésima en el nivel i, $\forall i \in \{1, \cdots, j-1\}$, $t_j > 0$ y es menor o igual que el número de reordenaciones factibles en el nivel j (ó $t_j = 0$ si no existen reordenaciones factibles en dicho nivel).

Dada una solución α-factible del DTSPMS, un trazado $(t_1, \cdots, t_\alpha)$ de tamaño α determina una ejecución del algoritmo MSR para reducir en α unidades el tamaño máximo de las pilas de la solución de partida, suponiendo que las reorganizaciones factibles que se pueden elegir en cada nivel están ordenadas de alguna manera. Dicha ejecución queda definida de la siguiente forma: para cada $j = 1, \cdots, \alpha$, el algoritmo elige la configuración factible t_j-ésima en el nivel j-ésimo, y si en algún caso $t_j = 0$ esto significa que no existen configuraciones factibles en el nivel j. En el Ejemplo 2 se ilustra cómo funciona un trazado sencillo.

Ejemplo 2. Sea $t = (3, 1, 4)$ un trazado de tamaño 3 para una solución S en la que el tamaño máximo de pilas es 7, pero la capacidad máxima de las pilas en la instancia a resolver es $Q = 4$. El trazado t indica que en el primer nivel, para pasar de 7 unidades de tamaño máximo a 6, se utilizará la tercera recolocación factible, en el segundo nivel (5 unidades de tamaño máximo) la primera y en el tercer nivel (4 unidades de tamaño máximo) la cuarta. Implementando dichas recolocaciones en el orden dado por el trazado se obtendrá una solución factible.

Definición 36. Sea $S \in X_\alpha$ una solución α-factible.

i) Los conjuntos $X^\alpha(S)$ e $Y^\alpha(S)$ están formados por todas las reorganizaciones consideradas por los algoritmos MSRB y GRAS, respectivamente, al aplicarse a la solución S para reducir α unidades el tamaño máximo de las pilas.

ii) Sea $t = (t_1, \cdots, t_\alpha)$ un trazado de tamaño α para S. El conjunto $Z^t(S)$ consta de todas las reorganizaciones consideradas por el algoritmo MSR al aplicarse a S para reducir α unidades el tamaño máximo de las pilas siguiendo la ejecución determinada por t.

Los conjuntos $X^\alpha(S)$ e $Y^\alpha(S)$ representan las reorganizaciones consideradas por los algoritmos determinísticos (MSRB y GRAS), mientras que el conjunto $Z^t(S)$ representa las del algoritmo MSR, que no es determinístico y por tanto requiere la elección de un trazado t para su definición.

Observación 36. Un elemento de $Y^\alpha(S)$ viene dado por una aplicación $\xi : C \longrightarrow P_p$ que indica que cada encargo cabecero $c \in C$ se reasigna a la pila $\xi(c) \in P_p$. Una reorganización perteneciente a los conjuntos $X^\alpha(S)$ ó $Z^t(S)$ según la cual se reduce el tamaño máximo de las pilas de la solución en α unidades viene determinada por α aplicaciones $\xi_1, \cdots, \xi_\alpha$ de la forma $\xi_j : C^j \longrightarrow P_p^j$, $\forall j = 1, \cdots, \alpha$, de manera que cada encargo $c \in C^j$ de cada nivel $j \in \{1, \cdots, \alpha\}$ se reasigna a la pila $\xi_j(c)$. Así, los elementos de $Y^\alpha(S)$ se denotarán como $\xi \in Y^\alpha(S)$ y los de $X^\alpha(S)$ y $Z^t(S)$ como $(\xi_1, \cdots, \xi_\alpha) \in X^\alpha(S)$ ó $Z^t(S)$.

Definición 37. Sea $S \in X_\alpha$ una solución α-factible y $t \in \mathbb{N}^\alpha$ un trazado para S. Realizando un pequeño abuso de notación, para cada $W \in \{X^\alpha(S), Z^t(S)\}$ se dice que:

i) $\xi \in Y^\alpha(S)$ *pertenece a* W, y se denota por $\xi \in W$, si existe $(\xi_1, \cdots, \xi_\alpha) \in W$ tal que la asignación de pilas que se obtiene al aplicar $(\xi_1, \cdots, \xi_\alpha)$ a S es la misma que se obtiene al aplicar ξ. En caso afirmativo, se dice que $(\xi_1, \cdots, \xi_\alpha)$ es una *reorganización por niveles* asociada a la *reorganización global* ξ.

ii) $(\xi_1, \cdots, \xi_\alpha) \in W$ *pertenece a* $Y^\alpha(S)$, y se denota por $(\xi_1, \cdots, \xi_\alpha) \in Y^\alpha(S)$, si existe $\xi \in Y^\alpha(S)$ tal que la asignación de pilas que se obtiene al aplicar ξ a S es la misma que

se obtiene al aplicar $(\xi_1, \cdots, \xi_\alpha)$. En caso afirmativo, se dice que ξ es la *reorganización global* asociada a la *reorganización por niveles* $(\xi_1, \cdots, \xi_\alpha)$.

En el Ejemplo 3 se ilustra la definición 37.

Ejemplo 3. Sea una instancia del DTSPMS con $n = 6$ encargos y $m = 3$ pilas con $Q = 2$ unidades de capacidad, y sea $\alpha = 2$ el margen utilizado en las soluciones α-factibles. Consideremos una solución 2-factible $S = (\pi_1, \pi_2, \lambda)$ tal que $\pi_1 \equiv$ [0-1-2-3-4-5-6-0], $\pi_2 \equiv$ [0-6-4-5-2-3-1-0] y $\lambda(\{1,3,5,6\}) = 1$, $\lambda(2) = 2$, $\lambda(4) = 3$, que da lugar a una estructura de pilas como la que se muestra en la primera secuencia de las Figuras 5.1 y 5.2. Sean ξ_1 la reorganización de primer nivel que reasigna el encargo 6 a la pila 3 (segunda secuencia de la Figura 5.1), ξ_2 la reorganización de segundo nivel que reasigna el encargo 5 a la pila 2 (tercera secuencia de la Figura 5.1) y ξ la reorganización global que reasigna el encargo 6 a la pila 3 y el 5 a la 2 (segunda secuencia de la Figura 5.2). Entonces (ξ_1, ξ_2) es una reorganización por niveles asociada a la reorganización global ξ y viceversa, ξ es la reorganización global asociada a la reorganización por niveles (ξ_1, ξ_2), ya que la solución final obtenida tras aplicar ambas es la misma.

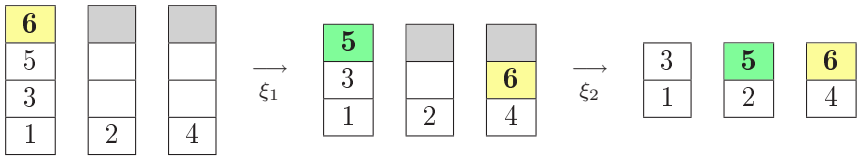


Figura 5.1: Reorganización por niveles (ξ_1, ξ_2) del Ejemplo 3

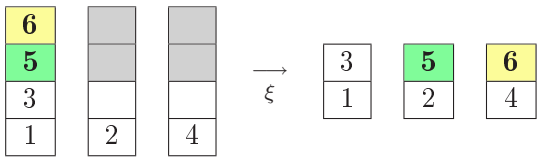


Figura 5.2: Reorganización global ξ del Ejemplo 3

Los conjuntos $X^\alpha(S)$, $Y^\alpha(S)$, $Z^t(S)$ representan las reorganizaciones que tienen en cuenta los algoritmos de reducción de pilas considerados, MSRB, GRAS y MSR. En las proposiciones que siguen se determinará la relación existente entre dichos conjuntos, mostrando la relación entre los algoritmos correspondientes.

Proposición 43. *Sea $S \in X_\alpha$ una solución α-factible y $T = \{t \in \mathbb{N}^\alpha \mid t$ trazado para $S\}$ el conjunto de trazados de tamaño α para S. Entonces $X^\alpha(S) = \bigcup_{t \in T} Z^t(S)$.*

Demostración

El resultado se obtiene fácilmente al observar que el algoritmo MSRB generaliza al algoritmo MSR, considerando todas las posibles elecciones que el algoritmo MSR puede realizar en cada nivel.

$\boxed{\subseteq}$ Sea $\bar{\xi} = (\xi_1, \cdots, \xi_\alpha) \in X^\alpha(S)$. Definamos un trazado $t = (t_1, \cdots, t_\alpha) \in \mathbb{N}^\alpha$ para S que determine que el algoritmo MSR elija la reorganización ξ_j en el nivel j para cada $j = 1, \cdots, \alpha$. Entonces, $\bar{\xi} \in Z^t(S)$, y por tanto $X^\alpha(S) \subseteq \bigcup_{t \in T} Z^t(S)$.

$\boxed{\supseteq}$ Sean $t \in T$ y $\bar{\xi} = (\xi_1, \cdots, \xi_\alpha) \in Z^t(S)$. Es claro que $\bar{\xi} \in X^\alpha(S)$, ya que el algoritmo MSRB considera cualquier elección en cada nivel, incluyendo la dada por el trazado t. Así, $X^\alpha(S) \supseteq \bigcup_{t \in T} Z^t(S)$. □

Observación 37. Sea $S \in X_\alpha$ una solución α-factible y $T = \{t \in \mathbb{N}^\alpha \mid t \text{ trazado para } S\}$. Entonces $Z^t(S) \subset X^\alpha(S)\ \forall t \in T$.

Proposición 44. *Si $S \in X_\alpha$ es una solución α-factible entonces $Y^\alpha(S) \subset X^\alpha(S)$.*

Demostración

Sea $S \in X_\alpha$ y $\xi \in Y^\alpha(S)$ una reorganización encontrada por el algoritmo GRAS según la cual el tamaño máximo de las pilas de S se reduce en α unidades. Sea q^* el tamaño máximo de las pilas de S y $q = q^* - \alpha$. Para probar que $\xi \in X^\alpha(S)$ hay que demostrar que existe una reorganización por niveles $\bar{\xi} = (\xi_1, \cdots, \xi_\alpha) \in X^\alpha(S)$ asociada a ξ.

Para cada $j \in \{1, \cdots, \alpha\}$ basta tomar

$$C^j = \{c \in D \mid c \text{ ocupa la posición } q - j + 1 \text{ en su pila de la solución } S\}$$

$$P_p^j = \{k \in P \mid |\lambda^{-1}(k)| < q^*\}$$

$$\begin{array}{rcl} \xi_j : C^j & \longrightarrow & P_p^j \\ c & \longmapsto & \xi_j(c) = \xi(c) \end{array}$$

El efecto que produce $\bar{\xi} = (\xi_1, \cdots, \xi_\alpha)$ sobre las pilas de S al aplicar secuencialmente cada ξ_j es exactamente el mismo que produce ξ, ya que las reasignaciones de pilas, aunque pasan a realizarse nivel por nivel, se mantienen. La construcción de esta nueva reorganización $\bar{\xi} \in X^\alpha(S)$ a partir de $\xi \in Y^\alpha(S)$ se puede llevar a cabo gracias a que el tamaño de las pilas a las que se reasignan los encargos cabeceros en ξ es menor o igual que q^*, de manera que también se pueden reasignar encargos a estas pilas en cualquiera de los niveles de $\bar{\xi}$. Cuando en alguno de los niveles una de estas pilas se llena, está garantizado que no se le va

a reasignar ningún encargo más en niveles posteriores, ya que en tal caso la reorganización de partida ξ resultaría ser no factible. □

El recíproco no es cierto, ya que existen reordenaciones por niveles del conjunto $X^\alpha(S)$ que no tienen una reordenación global asociada en el conjunto $Y^\alpha(S)$, como se verá a continuación.

Proposición 45. *Sea $S \in X_\alpha$. Entonces para cada reorganización global $\xi \in Y^\alpha(S)$ existe un trazado t de tamaño α para S tal que $\xi \in Z^t(S)$.*

Demostración

Es consecuencia inmediata de las proposiciones 43 y 44, pues al ser $\xi \in Y^\alpha(S) \subset X^\alpha(S) = \bigcup_{t \in T} Z^t(S)$ entonces existe $t \in T$ tal que $\xi \in Z^t(S)$. □

Proposición 46. *Para cada una de las relaciones i)-iii) existe una instancia del DTSPMS, un margen $\alpha \in \mathbb{N}$, una solución α-potencial $S \in X_\alpha$ y un trazado $t = (t_1, \cdots, t_\alpha)$ de tamaño α para S que hace que dicha relación se verifique, donde*

i) $Z^t(S) \nsubseteq Y^\alpha(S)$.

ii) $Y^\alpha(S) \nsubseteq Z^t(S)$.

iii) $X^\alpha(S) \nsubseteq Y^\alpha(S) \cup Z^t(S)$.

Demostración

i) Consideremos una instancia del DTSPMS con un conjunto de encargos $D = \{1, 2, \cdots, 9\}$, 3 pilas disponibles y un tamaño máximo por pila de $Q = 3$ unidades. Definamos la solución $S = (\pi_1, \pi_2, \lambda)$ de la siguiente manera:

$$\begin{array}{ll} \pi_1(i) = i & \forall i \in D \\ \pi_2(i) = 10 - i & \forall i \in D \end{array} \qquad \lambda(c) = \begin{cases} 1 & \text{si } c \in \{1, 2, 5, 7, 8\} \\ 2 & \text{si } c \in \{3\} \\ 3 & \text{si } c \in \{4, 6, 9\} \end{cases}$$

S es una solución del problema en la que el tamaño máximo de las pilas es $q = 5$, y por tanto es 2-factible. En la Figura 5.3 aparecen representados los elementos de dicha solución.

Para transformar S en una solución factible habría que modificar λ para reducir el tamaño de las pilas en $\alpha = 2$ unidades. Para ello definamos (ξ_1, ξ_2) de la siguiente forma:

$$P_g^1 = \{1\}, \quad P_p^1 = \{2, 3\}, \quad C^1 = \{8\}$$

Ruta 1 (recogida): 0-1-2-3-4-5-6-7-8-9-0
Ruta 2 (entrega): 0-9-8-7-6-5-4-3-2-1-0

Pilas:

8		
7		
5		9
2		6
1	3	4

Figura 5.3: Representación de la solución 2-factible S de *i)*

$$P_g^2 = \{1, 3\}, \quad P_p^2 = \{2\}, \quad C^2 = \{7, 9\}$$

$$\begin{array}{rcl} \xi_1 : C^1 & \longrightarrow & P_p^1 \\ 8 & \longmapsto & \xi_1(8) = 3 \end{array} \qquad \begin{array}{rcl} \xi_2 : C^2 & \longrightarrow & P_p^2 \\ 7 & \longmapsto & \xi_2(7) = 2 \\ 9 & \longmapsto & \xi_2(9) = 2 \end{array}$$

Para cierto trazado $t \in \mathbb{N}^2$ según el cual se elige ξ_1 en el primer nivel y ξ_2 en el segundo nivel, se tiene que $(\xi_1, \xi_2) \in Z^t(S)$. Al reorganizar las pilas de S según (ξ_1, ξ_2) se obtiene una nueva asignación de pilas, denotada por $\hat{\lambda}$, que queda definida como sigue:

$$\hat{\lambda}(c) = \begin{cases} 1 & \text{si } c \in \{1, 2, 5\} \\ 2 & \text{si } c \in \{3, 7, 9\} \\ 3 & \text{si } c \in \{4, 6, 8\} \end{cases} \quad \forall c \in D$$

En la Figura 5.4 se observa cómo se obtiene $\hat{\lambda}$ modificando la asignación de pilas aplicando consecutivamente las reorganizaciones ξ_1 y ξ_2. Esta reorganización, que consigue disminuir el tamaño de las pilas hasta $Q = 3$, no es considerada, sin embargo, por el algoritmo GRAS al ser aplicado a S con $\alpha^* = 0$, ya que la pila 3 tiene tamaño exactamente 3, por lo que no dispone de espacio libre para albergar más encargos ni tiene encargos excedentes, y por tanto no es considerada en ninguna reorganización global del algoritmo GRAS; así, el encargo 9, asignado a la pila 3, nunca podría ser reasignado, cosa que sí ocurre en la reorganización dada por (ξ_1, ξ_2).

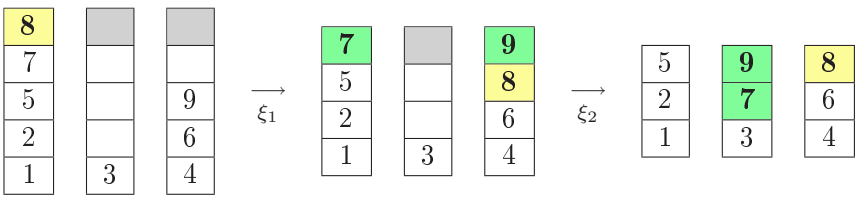


Figura 5.4: Reorganización de las pilas de S según (ξ_1, ξ_2)

Consecuentemente no existe ninguna reordenación global asociada a (ξ_1, ξ_2), por lo que

$(\xi_1, \xi_2) \notin Y^\alpha(S)$, y por tanto se tiene que $Z^t(S) \not\subseteq Y^\alpha(S)$.

ii) Consideremos la misma instancia del DTSPMS del apartado anterior y definamos la solución $S = (\pi_1, \pi_2, \lambda)$ como $\pi_1 \equiv [0,1,2,3,5,6,7,8,4,9,0]$, $\pi_2 \equiv [0,9,4,7,8,6,5,3,2,1,0]$ y

$$\lambda(c) = \begin{cases} 1 & \text{si } c \in \{1,2,3,7,4\} \\ 2 & \text{si } c \in \{5,6\} \\ 3 & \text{si } c \in \{8,9\} \end{cases} \qquad \forall c \in D$$

S es una solución del problema en la que el tamaño máximo de las pilas es 5, y por tanto también es 2-factible. En la Figura 5.5 aparecen representados los elementos de dicha solución.

Ruta 1 (recogida): 0-1-2-3-5-6-7-8-4-9-0
Ruta 2 (entrega): 0-9-4-7-8-6-5-3-2-1-0

Pilas:

4		
7		
3		
2	6	9
1	5	8

Figura 5.5: Representación de la solución 2-factible S de de *ii)*

Para transformar S en una solución factible habría que modificar λ para reducir el tamaño de las pilas en $\alpha = 2$ unidades. Para ello definimos $\xi \in Y^\alpha(S)$ de la siguiente forma:

$$\begin{array}{rcl} \xi : C & \longrightarrow & P_p \\ 4 & \longmapsto & \xi(4) = 3 \\ 7 & \longmapsto & \xi(7) = 2 \end{array} \quad \text{donde} \quad P_g = \{1\}, \quad P_p = \{2,3\}, \quad C = \{4,7\}.$$

El efecto que produce la reorganización global ξ sobre las pilas de S queda representado en la Figura 5.6.

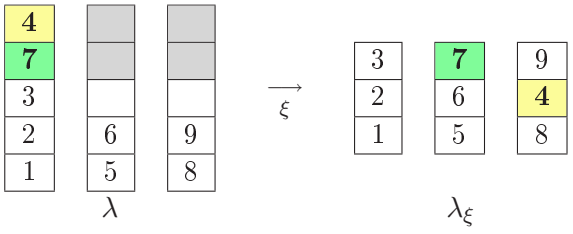


Figura 5.6: Reorganización de las pilas de S según ξ

Ahora veremos que existe un trazado $t = (t_1, t_2) \in \mathbb{N}^2$ para S tal que $\xi \notin Z^t(S)$. En el

primer nivel (descender el tamaño de las pilas 1 unidad) el único encargo a recolocar es 4, que puede reasignarse tanto a la pila 2 como a la pila 3. Si se elige reasignar dicho encargo a la pila 2 (denotemos dicha reorganización por ξ_1), la asignación de pilas que se obtiene, denotada por λ_1, es la que se presenta en la Figura 5.7. Definamos t_1 de manera que en el primer nivel del algoritmo MSR aplicado a S se elija la reorganización ξ_1.

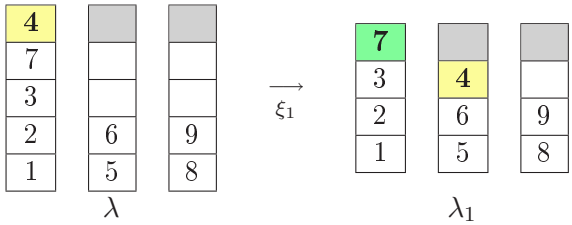


Figura 5.7: Reorganización de las pilas de S según ξ_1

A partir de la nueva asignación de pilas λ_1 el algoritmo MSR pasa al siguiente nivel, en el que el único encargo por recolocar es 7 y la única pila con espacio disponible es la pila 3. Sin embargo, al colocar el encargo 7 en cualquier posición de la pila 3 se violan las condiciones de precedencia impuestas por las rutas π_1 y π_2, por lo que no existe ninguna reorganización factible y el algoritmo debe parar sin conseguir una solución factible. Por tanto, la segunda componente del trazado t es $t_2 = 0$.

Para haber podido alcanzar la única reorganización factible, que es ξ, la primera componente del trazado debería ser distinta de t_1, ya que de esta forma el algoritmo MSR podría haber reasignado el encargo 4 a la pila 3 en el primer nivel para así poder recolocar el encargo 7 en la pila 2 en el segundo nivel.

Así, se tiene que existe un trazado $t \in \mathbb{N}^2$ tal que $\xi \notin Z^t(S)$, de donde se deduce que para dicho trazado $Y^\alpha(S) \nsubseteq Z^t(S)$.

iii) Sean $\xi = (\xi_1, \xi_2)$ y t, respectivamente, la reorganización y el trazado definidos en el apartado *i)*. Según se demuestra en dicho apartado es $\xi \notin Y^\alpha(S)$ y $\xi \in Z^t(S)$, y como $Z^t(S) \subseteq X^\alpha(S)$, también es $\xi \in X^\alpha(S)$.

Por otro lado, sea $\hat{t} \in \mathbb{N}^\alpha$ el trazado de S según el cual en el primer nivel se recoloca el encargo 8 en la pila 2 y en el segundo se recoloca el encargo 7 también en la pila 2. Con esta elección se obtiene que $\xi \notin Z^{\hat{t}}(S)$. Por tanto $\xi \in X^\alpha(S)$, pero $\xi \notin Y^\alpha(S)$ y $\xi \notin Z^{\hat{t}}(S)$, de donde se deduce que $X^\alpha(S) \nsubseteq Y^\alpha(S) \cup Z^{\hat{t}}(S)$. □

En la Figura 5.8 se presenta un esquema en el que se muestran las relaciones de contenido que existen entre los conjuntos de reorganizaciones que exploran los distintos algoritmos de reducción de pilas y que fueron introducidas en las proposiciones anteriores. El algoritmo

MSRB generaliza los algoritmos MSR y GRAS, explorando siempre todas las reorganizaciones consideradas por ellos, pero su complejidad es mucho mayor. Entre los algoritmos MSR y GRAS, sin embargo, no se verifica ninguna relación de contenido, de manera que pueden utilizarse como algoritmos complementarios y serán los que habitualmente consideraremos en la práctica.

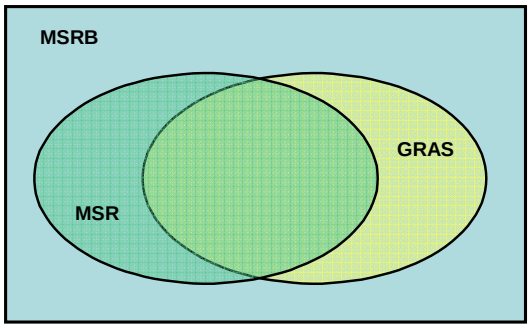


Figura 5.8: Relaciones de contenido entre los algoritmos de reducción de pilas

Nótese que, aun explorando todas las reorganizaciones posibles, ninguno de estos algoritmos garantiza la obtención de una solución factible que verifique las restricciones de capacidad.

5.2. Infactibilidad por Precedencias: soluciones potenciales

Las condiciones de precedencia del DTSPMS determinan la relación entre las rutas de recogida y entrega de las soluciones, reduciendo en gran medida el número de parejas de rutas factibles y restringiendo enormemente el proceso de búsqueda. Los operadores que determinan las estructuras de entornos para soluciones factibles introducidas en la Sección 4.3 se definen a través de movimientos cuya complejidad se debe en gran medida a la necesidad de mantener la factibilidad de la solución inicial, sobre todo respecto a las restricciones de precedencia. Sin embargo, los operadores introducidos en la Sección 4.4, que permiten el incumplimiento de algunas restricciones de precedencia, son más sencillos y se pueden implementar y ejecutar de forma más rápida. Estos operadores dan lugar a un proceso de búsqueda más dinámico, que se mueve por el espacio de soluciones infactibles del problema y que, dirigido convenientemente, puede permitir la obtención de buenas soluciones factibles.

Al permitir el uso de movimientos que violen algunas restricciones de precedencia se introduce una infactibilidad en la búsqueda que denominaremos *Infactibilidad por Precedencias* (IP), dando lugar a unas soluciones que llamaremos soluciones potenciales (definición 21). Estas soluciones, al igual que las soluciones α-factibles, se utilizarán como soluciones

intermedias en algunas heurísticas, pero sólo de forma temporal para flexibilizar el procedimiento de búsqueda, pues todas estas heurísticas devolverán siempre como resultado final una solución factible (0-factible).

De la misma manera que ocurría para la infactibilidad IC, para manejar adecuadamente la infactibilidad IP es necesario medirla correctamente y diseñar procedimientos con los cuales se puedan recolocar los encargos que violen alguna restricción de precedencia, permitiendo la obtención de soluciones factibles a partir de soluciones potenciales. En las siguientes secciones se presentan varias medidas para la infactibilidad IP y se detallan diversos procedimientos para eliminarla.

5.2.1. Medidas de la Infactibilidad por Precedencias

Una solución potencial es no factible si viola alguna de las restricciones de precedencia impuestas por el orden LIFO a seguir en cada pila. Algunas soluciones potenciales pueden transformarse en factibles realizando cambios mínimos que apenas modifican el valor de la función objetivo, mientras que otras requieren una modificación más profunda que altera en gran medida el valor de la función objetivo. Al tratar con soluciones potenciales es crucial poder medir de una manera razonable e informativa cuán infactibles son, para así poder evaluar a qué distancia se encuentran de la factibilidad. Para ello se introducen varias *medidas de infactiblidad* que representan de distintas formas el grado de infactibilidad IP de una solución potencial dada.

La conversión de una solución potencial en factible será más complicada cuanto mayor sea el número de restricciones de precedencia que se violan, y de hecho la solución será factible si el número de restricciones de precedencia violadas es cero; por tanto, tomaremos el número de restricciones de precedencia violadas como una medida de la infactibilidad IP.

Según el Modelo 1 (2.11)-(2.20), las restricciones de precedencia del problema son de la forma

$$y_{ij}^1 + z_{ip} + z_{jp} \leq 3 - y_{ij}^2 \qquad \forall i, j \in V_*^\delta, \forall p \in P \tag{5.6}$$

De ellas sólo son no redundantes aquellas correspondientes a parejas de encargos asignados a la misma pila. Para cada pareja de encargos $i, j \in V_*^\delta$ asignados a una misma pila $p \in P$, la desigualdad (5.6) se verificará si y sólo si dichos encargos se recogen y entregan en orden inverso. Así, el número de restricciones de precedencia violadas coincide con el número de parejas de encargos asignados a la misma pila que se entregan y recogen en el mismo orden, que se llamarán *parejas descolocadas*.

En la implementación de algoritmos heurísticos no se trabaja con variables binarias, sino que las soluciones se representan con una terna de aplicaciones de la forma (π_1, π_2, λ),

por lo que contar el número de restricciones del tipo (5.6) violadas es complicado; así, lo que se hace es contar el número de parejas descolocadas, que es equivalente y más eficiente computacionalmente. La definición de la medida de infactibilidad IP obtenida de esta manera se precisa a continuación.

Definición 38. La aplicación $\Lambda_P^1 : \tilde{X} \longrightarrow \mathbb{N} \cup \{0\}$ asigna a cada solución potencial un número entero no negativo que representa su infactibilidad IP. Para cada $S = (\pi_1, \pi_2, \lambda) \in \tilde{X}$ la medida $\Lambda_P^1(S)$, se define como sigue:

$$\Lambda_P^1(S) = \text{card}\{(i,j) \in D^2 \,|\, \lambda(i) = \lambda(j),\, \pi_1^{-1}(i) < \pi_1^{-1}(j), \pi_2^{-1}(i) < \pi_2^{-1}(j)\} \tag{5.7}$$

Disponiendo exclusivamente de las dos rutas de la solución y la asignación de pilas de una solución potencial $(\pi_1, \pi_2, \lambda) \in \tilde{X}$, son necesarias muchas operaciones para contar cuántas parejas se recogen y entregan en el mismo orden. Para aliviar este gasto computacional se utiliza una matriz $\Theta = (\theta_{ij})_{i,j \in D}$ que representa las relaciones de precedencia de una manera más adecuada para el cálculo de la medida Λ_P^1.

Definición 39. Sea $S = (\pi_1, \pi_2, \lambda) \in \tilde{X}$ una solución potencial. La *matriz de precedencias* $\Theta = (\theta_{ij})_{i,j \in D}$ de S se define de la siguiente manera:

$$\theta_{ij} = \begin{cases} 1 & \text{si } \pi_1^{-1}(i) < \pi_1^{-1}(j),\ \lambda(i) = \lambda(j) \\ -1 & \text{si } \pi_1^{-1}(i) > \pi_1^{-1}(j),\ \lambda(i) = \lambda(j) \\ 0 & \text{si } \lambda(i) \neq \lambda(j) \end{cases}$$

$$\theta_{ji} = \begin{cases} 1 & \text{si } \pi_2^{-1}(i) < \pi_2^{-1}(j),\ \lambda(i) = \lambda(j) \\ -1 & \text{si } \pi_2^{-1}(i) > \pi_2^{-1}(j),\ \lambda(i) = \lambda(j) \\ 0 & \text{si } \lambda(i) \neq \lambda(j) \end{cases} \qquad \forall i, j \in D,\ i < j$$

$$\theta_{ii} = 0 \ \forall i \in D$$

Esta matriz se construye para la solución inicial considerada y posteriormente se actualiza a medida que la solución se va modificando. La construcción inicial de la matriz requiere el mayor coste computacional, ya que la actualización de la matriz es sencilla.

A partir de la matriz de precedencias $\Theta = (\theta_{ij})_{i,j \in D}$ de una solución S se puede calcular fácilmente la medida $\Lambda_P^1(S)$, según indica la proposición 47.

Proposición 47. *Sea $\Theta = (\theta_{ij})_{i,j \in D}$ la matriz de precedencias de una solución potencial $S = (\pi_1, \pi_2, \lambda) \in \tilde{X}$ introducida en la definición 39. Se verifica que*

$$\Lambda_P^1(S) = \frac{1}{2} \sum_{i \in D} \sum_{j > i} |\theta_{ij} + \theta_{ji}| \tag{5.8}$$

Demostración

Sea $(i,j) \in D^2$ una pareja de encargos tal que $i < j$ y $\lambda(i) = \lambda(j)$. Estará descolocada si

$$\begin{array}{ccc} \pi_1^{-1}(i) < \pi_1^{-1}(j)\ [\Leftrightarrow \theta_{ij} = 1] & , & \pi_2^{-1}(i) < \pi_2^{-1}(j)\ [\Leftrightarrow \theta_{ji} = 1] \\ & \text{ó} & \\ \pi_1^{-1}(i) > \pi_1^{-1}(j)\ [\Leftrightarrow \theta_{ij} = -1] & , & \pi_2^{-1}(i) > \pi_2^{-1}(j)\ [\Leftrightarrow \theta_{ji} = -1] \end{array}$$

lo cual es equivalente a que el producto $(\pi_1^{-1}(j) - \pi_1^{-1}(i)) \cdot (\pi_2^{-1}(j) - \pi_2^{-1}(i))$ tenga signo positivo, y también es equivalente a que $[\theta_{ij} = 1, \theta_{ji} = 1]$, ó $[\theta_{ij} = -1, \theta_{ji} = -1]$. Ahora bien, esta última condición se da si y sólo si $\theta_{ij} + \theta_{ji} \in \{2, -2\}$, o lo que es lo mismo, $|\theta_{ij} + \theta_{ji}| = 2$.

Así, se tiene que:

$$\begin{array}{c} \Lambda_P^1(S) = \text{card}\{(i,j) \in D^2 \,|\, \lambda(i) = \lambda(j),\, \pi_1^{-1}(i) < \pi_1^{-1}(j), \pi_2^{-1}(i) < \pi_2^{-1}(j)\} = \\ = \text{card}\{(i,j) \in D^2 \,|\, i < j,\, |\theta_{ij} + \theta_{ji}| = 2\} = \sum_{(i,j) \in D^2, i<j} \frac{1}{2} |\theta_{ij} + \theta_{ji}| = \\ = \frac{1}{2} \sum_{i \in D} \sum_{j>i} |\theta_{ij} + \theta_{ji}| \end{array}$$

como se quería demostrar. □

Además de la medida Λ_P^1 presentada anteriormente, hay otras medidas de la infactibilidad IP que surgen de manera natural. Fijado el número de parejas descolocadas, cuanto menor sea el número de pilas que contengan encargos pertenecientes a parejas descolocadas más sencillo será transformar la solución para conseguir que se verifiquen las restricciones de precedencia. Así, la infactibilidad IP será mayor cuantas más pilas tengan asignada alguna pareja descolocada, lo que motiva la definición de una nueva medida Λ_P^2. Además, si varias parejas descolocadas tienen un encargo común, las restricciones violadas correspondientes quizá puedan arreglarse moviendo sólo dicho encargo, por lo que las parejas disjuntas son las que más dificultan el cumplimiento de las restricciones de precedencia. Así, la infactibilidad IP será mayor cuantos más encargos distintos pertenezcan a parejas descolocadas, lo que motiva la definición de una nueva medida Λ_P^3. Las dos nuevas medidas propuestas se detallan en la definición 40.

Definición 40. Las aplicaciones $\Lambda_P^2, \Lambda_P^3 : \tilde{X} \longrightarrow \mathbb{N} \cup \{0\}$ asignan a cada solución potencial números enteros no negativos que representan distintas *medidas de su infactibilidad IP*. Para cada $S = (\pi_1, \pi_2, \lambda) \in \tilde{X}$, las medidas $\Lambda_P^j(S)$, $j = 2, 3$, se definen de la siguiente forma:

$$\Lambda_P^2(S) = \text{card}\{p \in P \,|\, \exists i, j \in \lambda^{-1}(p),\, (\pi_1^{-1}(j) - \pi_1^{-1}(i)) \cdot (\pi_2^{-1}(j) - \pi_2^{-1}(i)) > 0\}$$

$$\Lambda_P^3(S) = \text{card}\{i \in D \,|\, \exists j \in D,\, \lambda(i) = \lambda(j),\, (\pi_1^{-1}(j) - \pi_1^{-1}(i)) \cdot (\pi_2^{-1}(j) - \pi_2^{-1}(i)) > 0\}$$

El cálculo de las medidas Λ_P^2 y Λ_P^3 también es costoso computacionalmente, pero se simplifica utilizando la matriz de precedencias Θ, como se muestra en la proposicion 48.

Proposición 48. *Sea $\Theta = (\theta_{ij})_{i,j\in D}$ la matriz de precedencias de una solución potencial $S = (\pi_1, \pi_2, \lambda) \in \tilde{X}$ introducida en la definición 39, y para cada $i \in D$, $p \in P$ sean*

$$\hat{\theta}_i = \begin{cases} 1 & si \quad \sum_{j\in D} |\theta_{ij} + \theta_{ji}| > 0 \\ 0 & si \quad \sum_{j\in D} |\theta_{ij} + \theta_{ji}| = 0 \end{cases} \qquad \bar{\theta}_p = \begin{cases} 1 & si \quad \sum_{j\in \lambda^{-1}(p)} \hat{\theta}_j > 0 \\ 0 & si \quad \sum_{j\in \lambda^{-1}(p)} \hat{\theta}_j = 0 \end{cases}$$

Entonces se verifica que $\Lambda_P^3(S) = \sum_{i\in D} \hat{\theta}_i$ y $\Lambda_P^2(S) = \sum_{p\in P} \bar{\theta}_p$.

Demostración

Por definición, $\hat{\theta}_i = 1$ si y sólo si existe $j \in D$ tal que $|\theta_{ij} + \theta_{ji}| > 0$, es decir, si y sólo si existe $j \in D$ tal que i y j forman una *pareja descolocada*. Por tanto, $\sum_{i\in D} \hat{\theta}_i$ representa el número de encargos que pertenecen a parejas descolocadas, que es $\Lambda_P^3(S)$.

Por otro lado, también por definición se tiene que $\bar{\theta}_p = 1$ si y sólo si existe $j \in \lambda^{-1}(p)$ asignado a la pila p tal que $\hat{\theta}_j = 1$, es decir, si y sólo si existe un encargo asignado a la pila p perteneciente a alguna pareja descolocada. Por tanto, $\sum_{p\in P} \bar{\theta}_p$ representa el número de pilas con parejas descolocadas, que es $\Lambda_P^2(S)$. □

En el Ejemplo 4 que se presenta a continuación se ilustra el cálculo de la matriz de precedencias Θ y de las medidas de infactibilidad Λ_P^1, Λ_P^2 y Λ_P^3.

Ejemplo 4. Sea una instancia del DTSPMS con $n = 9$ encargos y $m = 3$ pilas con $Q = 3$ unidades de capacidad. Consideremos una solución potencial $S = (\pi_1, \pi_2, \lambda)$ tal que $\pi_1 \equiv$ [0-1-2-3-4-5-6-7-8-9-0], $\pi_2 \equiv$ [0-8-6-5-9-1-3-7-4-2-0] y $\lambda(\{1,2,3\}) = 1$, $\lambda(\{4,5,6\}) = 2$, $\lambda(\{7,8,9\}) = 3$. El plan de carga de S se puede representar de dos formas, según los encargos sean ordenados siguiendo la ruta de recogida (primer plan de la Figura 5.9, sobre fondo blanco) o la de entrega (segundo plan de la Figura 5.9, sobre fondo gris). En una solución factible ambos planes de carga coinciden, ya que los órdenes relativos de encargos asignados a la misma pila son inversos en las rutas de la solución, pero en una solución potencial este puede no ser el caso. Se tiene que $D = I = \{1, \cdots, 9\}$, $P = \{1,2,3\}$, $\pi_1, \pi_2 \in \text{Biy}(I, D)$ y $\lambda \in \text{Apl}(D, P)$. Las aplicaciones π_1, π_2 quedan definidas de la siguiente forma: $\pi_1(i) = i \ \forall i \in I$, $\pi_2(1) = 8$, $\pi_2(2) = 6$, $\pi_2(3) = 5$, $\pi_2(4) = 9$, $\pi_2(5) = 1$, $\pi_2(6) = 3$, $\pi_2(7) = 7$, $\pi_2(8) = 4$, $\pi_2(9) = 2$.

Figura 5.9: Distintos planes de carga de la solución S del Ejemplo 4

La matriz de precedencias Θ de S construida siguiendo la definición 39 es la siguiente:

$$\Theta = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 \end{pmatrix}$$

Siguiendo la definición 38, la medida de infactibilidad Λ_P^1 se calcula contando el número de parejas de encargos asignados a la misma pila que se recogen y entregan en el mismo orden, y que por tanto están colocados de distinta forma en los planes de carga de la Figura 5.9. Las parejas de encargos que cumplen lo anterior y que a consecuencia de ello no verifican las restricciones de precedencia son $\{1,2\}$, $\{1,3\}$, $\{8,9\}$. Así, $\Lambda_P^1 = 3$.

El valor de Λ_P^1 también se puede calcular a partir de Θ siguiendo la expresión dada en la proposición 47. Así, $\Lambda_P^1(S) = \frac{1}{2}\sum_{i\in D}\sum_{j>i}|\theta_{ij}+\theta_{ji}| = \frac{1}{2}(3|1+1|+6|1-1|+27|0+0|) = \frac{1}{2}\cdot 6 = 3$.

Calculemos ahora las medidas de infactibilidad Λ_P^2 y Λ_P^3 a partir de su definición (definición 40). Λ_P^2 es simplemente el número de pilas que contienen alguna pareja de encargos que violan las restricciones de precedencia, las cuales ya fueron identificadas anteriormente y son $\{1,2\}$, $\{1,3\}$, $\{8,9\}$. Estos encargos están asignados a las pilas 1 y 3, por lo que $\Lambda_P^2 = 2$. Por otro lado, para calcular Λ_P^3 hay que contar el número de encargos que aparecen en alguna de las parejas que violan las restricciones de precedencia. Los encargos que aparecen en dichas parejas son $\{1,2,3,8,9\}$, por lo que $\Lambda_P^3 = 5$.

Los valores de Λ_P^2 y Λ_P^3 también se puede calcular a partir de Θ siguiendo las expresiones dadas en la proposición 48. Se tiene que $\hat{\theta}_i = 1\ \forall i \in \{1,2,3,8,9\}$, $\hat{\theta}_i = 0\ \forall i \in \{4,5,6,7\}$, $\bar{\theta}_p = 1\ \forall p \in \{1,3\}$, $\bar{\theta}_2 = 0$. Así, $\Lambda_P^2(S) = \sum_{p\in P}\bar{\theta}_p = 1+0+1 = 2$ y $\Lambda_P^3(S) = \sum_{i\in D}\hat{\theta}_i = 5\cdot 1+4\cdot 0 = 5$, como ya habíamos calculado previamente.

5.2.2. Proyección simple de soluciones potenciales

La utilización de soluciones potenciales simplifica la realización de movimientos, la definición de estructuras de entornos y el proceso de búsqueda en general, ya que se relajan las restricciones de precedencia. Sin embargo, para conseguir soluciones factibles se necesita un mecanismo que permita obtenerlas a partir de soluciones potenciales, aunque para ello se aumente el coste de la solución. La transformación debe ser rápida y sencilla, consumiendo poco tiempo de cómputo y aumentando lo menos posible el coste total.

Las únicas restricciones posiblemente violadas por una solución potencial son las restricciones de precedencia impuestas por el principio LIFO de cada pila, es decir, una solución potencial es no factible si y sólo si existen parejas de encargos asignados a la misma pila que se recogen y entregan en el mismo orden. Así, si los encargos asignados a cada pila se reordenan de manera que los órdenes de recogida y entrega sean exactamente inversos, la solución que se obtiene verificará todas las restricciones de precedencia y por tanto será factible. Esta idea es la base para el mecanismo de transformación de soluciones potenciales descrito a continuación.

Definición 41. Sean $\delta \in \{1,2\}$ y $\gamma \in \{1,2\} \setminus \{\delta\}$. El *operador δ-proyección* $\Pi_\delta : \tilde{X} \longrightarrow X$ se define, para cada $(\pi_1, \pi_2, \lambda) \in \tilde{X}$, de la siguiente forma:

Para cada $k \in P$ sea $U_k = \{u_1^k, \cdots, u_{n_k}^k\} = \lambda^{-1}(k)$ el conjunto de encargos asignados a la pila k ordenados en orden inverso a como son atendidos en la ruta γ, esto es $\pi_\gamma^{-1}(u_1^k) > \cdots > \pi_\gamma^{-1}(u_{n_k}^k)$. Sea $P_k = \{p_1^k, \cdots, p_{n_k}^k\} = \pi_\delta^{-1}(U_k)$ las posiciones de los encargos de U_k en la ruta δ ordenadas de menor a mayor, es decir, $p_1^k < \cdots < p_{n_k}^k$. Entonces, $\Pi_\delta((\pi_1, \pi_2, \lambda)) = (\hat{\pi}_1, \hat{\pi}_2, \lambda)$, donde $\hat{\pi}_\delta(p_j^k) = u_j^k$, $\forall k \in P$, $\forall j = \{1, \cdots, n_k\}$ y $\hat{\pi}_\gamma = \pi_\gamma$.

Proposición 49. *El operador δ-proyección $\Pi_\delta : \tilde{X} \longrightarrow X$ está bien definido.*

Demostración

Sea $\gamma \in \{1,2\} \setminus \{\delta\}$. Para probar que $\Pi_\delta(S) \in X \ \forall S \in \tilde{X}$ basta comprobar que $\Pi_\delta(S)$ verifica las restricciones de precedencia, lo cual es obvio por la propia construcción de $\Pi_\delta(S)$, ya que los encargos asignados a cada pila son visitados en la ruta δ en orden inverso a como son visitados en la ruta γ. □

En el Ejemplo 5 se ilustra el funcionamiento del operador δ-proyección de la definición 41.

Ejemplo 5. Sea una instancia del DTSPMS con $n = 6$ encargos y $m = 2$ pilas con $Q = 3$ unidades de capacidad. Consideremos una solución potencial $S = (\pi_1, \pi_2, \lambda)$ tal que $\pi_1 \equiv$ [0-1-2-3-4-5-6-0], $\pi_2 \equiv$ [0-5-6-4-1-2-3-0] y $\lambda(\{1,2,4\}) = 1$, $\lambda(\{3,5,6\}) = 2$. El plan de carga

de S se puede representar de dos formas, según los encargos sean ordenados siguiendo la ruta de recogida (primer plan de la Figura 5.10, sobre fondo blanco) o la de entrega (segundo plan de la Figura 5.10, sobre fondo gris), y ambos pueden no coincidir al tratarse de una solución potencial.

Figura 5.10: Planes de carga de la solución potencial del Ejemplo 5

La idea del operador δ-proyección es realizar cambios en la ruta δ de manera que ambas planes de carga coincidan, imponiendo que en dicha ruta los encargos asignados a cada pila sigan un orden inverso al de la ruta que se mantiene fija. Por ejemplo, para realizar la 1-proyección de la solución S habría que modificar la ruta de recogida de manera que el primer plan de carga coincida con el segundo. Los encargos asignados a la primera pila se recogen siguiendo el orden 1-2-4, mientras que se entregan siguiendo el orden 4-1-2; ambas secuencias deberían ser opuestas para que la solución fuera factible, y como en este caso sólo permitimos modificar la ruta de recogida, el orden en dicha ruta debe ser 2-1-4. Considerando los encargos asignados a la segunda pila se tiene una situación similar: son recogidos siguiendo el orden 3-5-6 y entregados según 5-6-3, por lo que la primera secuencia debe cambiarse por 3-6-5. Combinando las dos nuevas secuencias se obtiene la ruta de recogida proyectada: $\hat{\pi}_1 \equiv$ [0-**2**-**1**-3-**4**-6-5]. Ahora la solución 1-proyectada $(\hat{\pi}_1, \pi_2, \lambda)$=$\Pi_1(S)$ es factible, y los dos posibles planes de carga coinciden (y son iguales al segundo plan de la Figura 5.10, sobre fondo gris).

El operador 2-proyección funciona de manera totalmente análoga, modificando la ruta de entrega y manteniendo fija la de recogida. Así, la secuencia 4-1-2 asociada a los encargos asignados a la primera pila debe cambiarse por la 4-2-1, y la 5-6-3 asociada a los de la segunda pila debe cambiarse por 6-5-3. Combinando ambas secuencias se obtiene la ruta de entrega proyectada: $\hat{\pi}_2 \equiv$ [0-6-5-**4**-**2**-**1**-3]. Al igual que antes, ahora la solución 2-proyectada $(\pi_1, \hat{\pi}_2, \lambda)$=$\Pi_2(S)$ es factible, y los dos posibles planes de carga coinciden (y son iguales al primer plan de la Figura 5.10, sobre fondo blanco).

El operador δ-proyección Π_δ sólo modifica la ruta δ de la solución, realizando las modificaciones necesarias sobre ella para que sea compatible con la otra ruta y la asignación de pilas. Se pueden considerar, por tanto, dos operadores, uno sobre cada ruta, e incluso un nuevo operador Π, definido en 42, que combina los dos anteriores y elige la mejor solución proyectada.

Definición 42. El *operador proyección* $\Pi : \tilde{X} \longrightarrow X$ se define de la siguiente forma:

$$\Pi(S) = \begin{cases} \Pi_1(S) & \text{si} \quad z(\Pi_1(S)) \leq z(\Pi_2(S)) \\ \\ \Pi_2(S) & \text{si} \quad z(\Pi_2(S)) < z(\Pi_1(S)) \end{cases} \qquad \forall S \in \tilde{X}$$

En el Ejemplo 5 se mostraba cómo obtener las dos soluciones proyectadas $(\hat{\pi}_1, \pi_2, \lambda) = \Pi_1(S)$ y $(\pi_1, \hat{\pi}_2, \lambda) = \Pi_2(S)$ a partir de la solución potencial S. El operador proyección Π introducido en la definición 42 consideraría ambas y elegiría aquella con menor coste total, mín$\{z(\hat{\pi}_1) + z(\pi_2),\ z(\pi_1) + z(\hat{\pi}_2)\}$.

Observación 38. Los operadores de proyección presentados anteriormente modifican una de las rutas de la solución para colocar las parejas que estén descolocadas y conseguir que la solución obtenida sea factible. Es posible que sean necesarios numerosos cambios para conseguir la factibilidad, y como consecuencia el coste de la solución aumente notablemente. Este efecto es difícil de controlar, aunque cuanto menor sea la medida de infactibilidad de la solución inicial menor será habitualmente el aumento del coste de su proyección.

5.2.3. Proyección optimizada de soluciones potenciales

El cálculo de la proyección simple de una solución potencial requiere pocas operaciones y se puede implementar de forma rápida y sencilla, ya que sólo considera una forma de recolocar los encargos que causan la infactibilidad IP. Se trata de un procedimiento rápido para obtener soluciones factibles a partir de soluciones potenciales, a costa de aumentar, en muchas ocasiones excesivamente, el coste de la solución de partida.

El objetivo del operador de proyección optimizada que se introduce a continuación es el mismo, recolocar los encargos que incumplen las restricciones de precedencia de manera que la solución obtenida sea factible, pero este nuevo operador examina distintas formas posibles de realizar dicha recolocación con el fin de minimizar el coste de la solución proyectada. Obviamente, el operador de proyección optimizada es más complejo y requiere más tiempo de cómputo para su ejecución.

Definición 43. Sean $A \subset D$ un subconjunto de encargos del problema, $I_A = \{1, 2, \cdots, n - |A|\}$, $\pi_1, \pi_2 \in \text{Biy}(I_A, D\backslash A)$ y $\lambda \in \text{Apl}(D\backslash A, P)$. La terna $S_A = (\pi_1, \pi_2, \lambda)$ se llama solución *A-parcial* del problema. Las permutaciones π_1, π_2 representan rutas de recogida y entrega para los encargos del conjunto $D\backslash A$ y λ una asignación de pilas para ellos, de manera que S_A sería una solución potencial para el DTSPMS asociado a los encargos $D\backslash A$. El conjunto de soluciones A-parciales se denotará por $\tilde{X}_A$.

Una solución A-parcial con $A = \varnothing$ se denominará solución parcial *completa*. Además,

si S_A respeta las restricciones de precedencia y el tamaño máximo de las pilas no es mayor que Q se dirá que S_A es una solución parcial factible.

Ejemplo 6. Consideremos el caso del Ejemplo 5, en el que se introducía una solución potencial $S = (\pi_1, \pi_2, \lambda)$ de una instancia del DTSPMS con 6 encargos y 2 pilas de capacidad 3. Se tiene que $D = I = \{1, \cdots, 6\}$, $P = \{1, 2\}$, $\pi_1, \pi_2 \in \text{Biy}(I, D)$ y $\lambda \in \text{Apl}(D, P)$. Los elementos de la solución S quedan definidos en términos de aplicaciones de la siguiente forma: $\pi_1 \equiv$[0,1,2,3,4,5,6,0], $\pi_2 \equiv$[0,5,6,4,1,2,3,0], $\lambda(c) = 1 \forall c \in \{1, 2, 4\}$, $\lambda(c) = 2$ $\forall c \in \{3, 5, 6\}$. Consideremos el subconjunto de encargos $A = \{1, 6\} \subset D$ y veamos cómo se construye la solución *A-parcial* $\tilde{S} = (\tilde{\pi}_1, \tilde{\pi}_2, \tilde{\lambda})$ asociada a S. Sus elementos $\tilde{\pi}_1, \tilde{\pi}_2 \in \text{Biy}(\{1, 2, 3, 4\}, \{2, 3, 4, 5\})$ y $\tilde{\lambda} \in \text{Apl}(\{2, 3, 4, 5\}, P)$ se definien a partir de la solución completa S, eliminando los encargos 1 y 6, de la siguiente forma: $\tilde{\pi}_1 \equiv$[0,2,3,4,5,0], $\tilde{\pi}_2 \equiv$[0,5,4,2,3,0], $\lambda(c) = 1$ $\forall c \in \{2, 4\}$, $\lambda(c) = 2$ $\forall c \in \{3, 5\}$. En la Figura 5.11 se presenta una representación esquemática de la solución $\tilde{S}$.

$\tilde{\pi}_1 \equiv$ [0-2-3-4-5-0]
$\tilde{\pi}_2 \equiv$ [0-5-4-2-3-0]

$\tilde{\lambda}$:

4	5
2	3

Figura 5.11: Solución $\{1, 6\}$-parcial $\tilde{S}$ del Ejemplo 6

La solución $\{1, 6\}$-parcial $\tilde{S}$, que sólo comprende 4 encargos, ahora sí verifica todas las restricciones de precedencia, por lo que es, de hecho, una solución parcial factible.

La definición de la medida de infactibilidad Λ_P^1 para soluciones potenciales completas puede extenderse fácilmente a soluciones parciales.

Definición 44. Sea $A \subset D$ un subconjunto de encargos. La medida de infactibilidad $\Lambda_P^1[A] : \tilde{X}_A \longrightarrow \mathbb{N} \cup \{0\}$ asigna a cada solución A-parcial un número entero no negativo que representa su infactibilidad IP. Para cada $S = (\pi_1, \pi_2, \lambda) \in \tilde{X}_A$, la medida $\Lambda_P^1[A](S)$ se define como sigue:

$$\Lambda_P^1[A](S) = \text{card}\{(i, j) \in (D \backslash A)^2 \,|\, \lambda(i) = \lambda(j),\, \pi_1^{-1}(i) < \pi_1^{-1}(j), \pi_2^{-1}(i) < \pi_2^{-1}(j)\}$$

Las soluciones parciales permiten representar soluciones en las cuales algunos encargos han sido eliminados temporalmente. Normalmente los encargos eliminados son aquellos que violan alguna restricción de precedencia y hacen que la solución dada sea no factible. Para distinguir este tipo de encargos es necesario medir la aportación de cada encargo a la infactibilidad IP de una solución potencial dada (parcial o completa), contando de forma independiente el número de restricciones violadas asociadas a cada encargo.

Definición 45. Sea $S = (\pi_1, \pi_2, \lambda) \in \tilde{X}_A$ una solución A-parcial. La medida de infactibilidad $\Lambda_P^1[A, S] : D \backslash A \longrightarrow \mathbb{N} \cup \{0\}$ asigna a cada encargo un número entero no negativo

que representa la *cantidad* de infactibilidad IP que aporta dicho encargo a la solución S. Para cada $u \in D \backslash A$, la medida $\Lambda_P^1[A,S](u)$ se define como sigue:

$$\Lambda_P^1[A,S](u) = \text{card}\{(u,j) \in (D\backslash A)^2 \,|\, \lambda(u) = \lambda(j),\ (\pi_1^{-1}(j) - \pi_1^{-1}(u)) \cdot (\pi_2^{-1}(j) - \pi_2^{-1}(u)) > 0\}$$

Definición 46. Sea S una solución potencial A-parcial. Se define el *conjunto de encargos infactibles* de S como $F_S^A = \{u \in D\backslash A \,|\, \Lambda_P^1[A,S](u) > 0\}$. Si S es no factible entonces $F_S^A \neq \varnothing$. Si S es completa entonces el conjunto de encargos infactibles $F_S^{\varnothing}$ se denota simplemente por F_S.

En el Ejemplo 7 se ilustra cómo se calcula la medida de infactibilidad $\Lambda_P^1[A,S]$ y el conjunto de encargos infactibles.

Ejemplo 7. Consideremos de nuevo el caso del Ejemplo 6 donde introducíamos una solución potencial completa S y una solución $\{1,6\}$-parcial $\tilde{S}$ para una instancia del DTSPMS con 6 encargos. Para la solución completa S se tiene que $\Lambda_P^1[\emptyset,S](1) = 1$, ya que el encargo 1 viola sólo una restricción de precedencia, pues se recoge y entrega antes que el 2, violando la restricción de precedencia correspondiente a la pareja 1-2, y se recoge antes que el 4 y se entrega después, verificando la restricción de precedencia correspondiente a la pareja 1-4. Análogamente, $\Lambda_P^1[\emptyset,S](2) = 1$, pues el encargo 2 es incompatible con el 1 pero no con el 4, y $\Lambda_P^1[\emptyset,S](4) = 0$, ya que el encargo 4 no viola ninguna restricción de precedencia. De la misma forma se calculan las medidas de infactibilidad de los encargos asignados a la pila 2: $\Lambda_P^1[\emptyset,S](3) = 0$, $\Lambda_P^1[\emptyset,S](5) = 1$, $\Lambda_P^1[\emptyset,S](6) = 1$. A partir de las medidas anteriores se construye fácilmente el *conjunto de encargos infactibles* de S, que está formado por los encargos con medida de infactibilidad positiva y que por tanto es $F_S = \{1,2,5,6\}$.

La solución $\{1,6\}$-parcial $\tilde{S}$ es factible, por lo que todos los encargos tienen medida de infactibilidad cero, $\Lambda_P^1[\{1,6\},\tilde{S}](c) = 0$, $\forall c \in \{2,3,4,5\}$, y el conjunto de encargos infactibles es vacío, $F_{\tilde{S}}^{\{1,6\}} = \varnothing$.

Si se eliminan todos los encargos de F_S^A, la solución parcial obtenida siempre será factible. Sin embargo, en la mayoría de los casos para obtener una solución parcial factible bastará con eliminar sólo unos cuantos elegidos convenientemente. En el Ejemplo 6 introducido anteriormente se observa claramente esta situación, ya que el conjunto de encargos infactibles de S es $F_S = \{1,2,5,6\}$ y sin embargo la solución $\{1,6\}$-parcial $\tilde{S}$ obtenida eliminando sólo los encargos 1 y 6 es factible.

Encontrar el menor subconjunto de F_S^A que dé lugar a una solución parcial factible es costoso computacionalmente, por lo que se utilizará una heurística greedy para encontrar subconjuntos lo más pequeños posible. El procedimiento greedy consiste en elegir de forma secuencial los encargos con medidas de infactibilidad $\Lambda_P^1[A,S]$ más altas hasta que no quede ninguno con medida de infactibilidad positiva. Evidentemente, cada vez que se elige

un encargo y se elimina de la solución se obtiene una nueva solución parcial, y por tanto la medida de infactibilidad del resto de encargos cambia.

Definición 47. Sean $A \subset D$ y $B \subset D \backslash A$. El operador $\Upsilon_B : \tilde{X}_A \longrightarrow \tilde{X}_{A \cup B}$ asigna a cada solución A-parcial S la solución $A \cup B$-parcial que se obtiene eliminando los encargos del conjunto B de las rutas de S y de su asignación de pilas.

Definición 48. Sea $S \in \tilde{X}$ una solución potencial completa y no factible. Sea $u_1 \in F_S$ tal que $\Lambda_P^1[\varnothing, S](u_1) \geq \Lambda_P^1[\varnothing, S](u),\ \forall u \in F_S$. Se definen recursivamente

$$S_0 = S,\ S_j = \Upsilon_{\{u_j\}}(S_{j-1})\ \forall j > 0$$

$$U_j = \{u_1, \cdots, u_j\}\ \forall j > 0$$

$$\forall j > 1,\ u_j \in F_{S_{j-1}} \text{ tal que } \Lambda_P^1[U_j, S_{j-1}](u_{j-1}) \geq \Lambda_P^1[U_{j-1}, S_{j-1}](u),\ \forall u \in F_{S_{j-1}}$$

A partir de ellos se definen también $j^* = \text{mín}\{j \in \mathbb{N} | F_{S_j} = \varnothing\}$, $B_S^* = \{u_1, \cdots, u_{j^*}\}$ y $B_S^o = (u_1, \cdots, u_{j^*})$. El conjunto B_S^* se denomina *conjunto de máxima infactibilidad IP de* S y la tupla ordenada B_S^o *vector de máxima infactibilidad IP de* S.

El conjunto $B_S^* \subset F_S$ contiene los encargos que aportan mayor infactibilidad, de manera que basta con eliminar dichos encargos para obtener una solución parcial factible, como indica la proposición 50. La tupla ordenada B_S^o contiene los mismos encargos que B_S^* pero ordenados decrecientemente según su infactibilidad.

El Ejemplo 8 ilustra cómo funciona el procedimiento recursivo descrito en la definición 48.

Ejemplo 8. Consideremos de nuevo la instancia del DTSPMS introducida en el Ejemplo 5 y calculemos $B_S^* \subset F_S$ y B_S^o para la solución potencial completa S descrita en dicho ejemplo. Como ya se determinó en el Ejemplo 7, el conjunto de encargos infactibles de S es $F_S = \{1, 2, 5, 6\}$ y sus medidas de infactibilidad son $\Lambda_P^1[\emptyset, S](c) = 1$, $\forall c \in F_S$. Siguiendo el procedimiento de la definición 48, $S_0 = S$ y u_1 podría ser cualquiera de los encargos de F_S, pues todos tienen la misma medida de infactibilidad; por ejemplo escojamos $u_1 = 1$, lo que implica que $U_1 = \{1\}$. Entonces $S_1 = \Upsilon_{\{1\}}(S_0)$, es decir, $S_1 = (\pi_1^1, \pi_2^1, \lambda^1)$ es la solución U_1-parcial obtenida eliminando el encargo 1 de S, y por tanto $\pi_1^1 \equiv$ [0-2-3-4-5-6-0], $\pi_2^1 \equiv$ [0-5-6-4-2-3-0] y $\lambda^1(c) = \lambda(c)$, $\forall c \in \{2, \cdots, 5\}$. Ahora debemos calcular $\Lambda_P^1[U_1, S_1](c)$, $\forall c \in D \backslash U_1$, que valen $\Lambda_P^1[U_1, S_1](c) = 0$, $\forall c \in \{2, 3, 4\}$ y $\Lambda_P^1[U_1, S_1](c) = 1$, $\forall c \in \{5, 6\}$, por lo que $F_{S_1} = \{5, 6\}$ y $u_2 = 6$ (también podría ser $u_2 = 5$). Entonces $U_2 = \{1, 6\}$ y $S_2 = \Upsilon_6(S_1) = \tilde{S}$, que es la solución $\tilde{S}$ ya calculada en el ejemplo de la Figura 5.11 y que ya sabemos que es una solución parcial factible. Por tanto, el proceso recursivo termina y se tiene que $B_S^* = \{u_1, u_2\} = \{1, 6\}$ y $B_S^o = (1, 6)$.

Proposición 50. *Sea $S = S_0 \in \tilde{X}$ una solución potencial completa y no factible y sea B_S^* el conjunto de máxima infactibilidad IP de S. Entonces $S^* = \Upsilon_{B_S^*}(S)$ es una solución B_S^*-parcial* factible*, es decir, $\Lambda_P^1[B_S^*](S^*) = 0$.*

Definición 49. Sean $A \subset D$, $A \neq \varnothing$, y $u \in A$. El operador $\dot{\Upsilon}_u : \tilde{X}_A \longrightarrow \tilde{X}_{A\backslash\{u\}}$ asigna a cada solución A-parcial S la solución $A\backslash\{u\}$-parcial $\dot{\Upsilon}_u(S)$ que se obtiene añadiendo el encargo u a la solución parcial S de la manera más conveniente; las posiciones de u en ambas rutas de $\dot{\Upsilon}_u(S)$ y la pila en la que se almacena se eligen de manera que $\Lambda_P^1[A\backslash\{u\}](\dot{\Upsilon}_u(S)) = \Lambda_P^1[A](S)$ (la infactibilidad IP no aumente) y el coste extra debido a la inclusión del nuevo encargo sea mínimo.

Ejemplo 9. Para ilustrar el funcionamiento del operador $\dot{\Upsilon}_u$ consideremos de nuevo la solución $\{1,6\}$-parcial $\tilde{S}$ del Ejemplo 6, que se obtiene a partir de S eliminando los encargos 1 y 6. Al aplicar el operador $\dot{\Upsilon}_1$ a la solución $\tilde{S} \in \tilde{X}_{\{1,6\}}$ se obtiene una nueva solución parcial $\dot{S} = (\dot{\pi}_1, \dot{\pi}_2, \dot{\lambda}) \in \tilde{X}_{\{6\}}$ con la misma medida de infactibilidad pero con un encargo más. Si $\dot{\lambda}(1) = 1$, las rutas $\dot{\pi}_1, \dot{\pi}_2$ podrían ser, por ejemplo, $\dot{\pi}_1 \equiv$ [0-1-2-3-4-5-0], $\dot{\pi}_2 \equiv$ [0-5-4-2-3-1-0], o también $\dot{\pi}_1 \equiv$ [0-2-3-4-1-5-0], $\dot{\pi}_2 \equiv$ [0-5-1-4-2-3-0], ya que así el encargo 1 no viola ninguna restricción de precedencia, pero no las rutas $\dot{\pi}_1 \equiv$ [0-1-2-3-4-5-0], $\dot{\pi}_2 \equiv$ [0-1-5-4-2-3-0], ya que añadirían nuevas infactibilidades a la solución. De entre todas las posibilidades válidas, a las que también habría que añadir las correspondientes al caso $\dot{\lambda}(1) = 2$, el operador $\dot{\Upsilon}_1$ elegiría aquella con menor coste total.

Definición 50. El *operador proyección optimizada* $\breve{\Pi} : \tilde{X} \longrightarrow X$ se define, para cada $S \in \tilde{X}$, de la siguiente forma:

$$\breve{\Pi}(S) = \dot{\Upsilon}_{u_{j^*}} \circ \cdots \circ \dot{\Upsilon}_{u_1} \circ \Upsilon_{u_{j^*}} \circ \cdots \circ \Upsilon_{u_1}(S) \tag{5.9}$$

donde $B_S^o = (u_1, \cdots, u_{j^*})$.

Proposición 51. *El operador $\breve{\Pi}$ está bien definido y por tanto la proyección optimizada de cualquier solución potencial es una solución completa y factible.*

Demostración

Al aplicar el operador compuesto $\Upsilon_{u_{j^*}} \circ \cdots \circ \Upsilon_{u_1}$ a S se obtiene una solución parcial factible, ya que $(u_1, \cdots, u_{j^*})$ es el vector de máxima infactibilidad IP de S. Posteriormente, al aplicar el operador $\dot{\Upsilon}_{u_{j^*}} \circ \cdots \circ \dot{\Upsilon}_{u_1}$ a la solución parcial obtenida, los encargos eliminados se vuelven a incluir en posiciones factibles, y por tanto la solución final obtenida es una solución completa y factible. □

Observación 39. El operador de proyección optimizada actúa eliminando el encargo que aporta mayor infactibilidad IP a la solución de partida y obtiene una solución parcial con infactibilidad menor; si esta solución parcial es infactible, se elimina de nuevo el encargo de mayor infactibilidad IP y se repite el proceso. Cuando se obtiene una solución parcial con infactibilidad IP nula, se vuelven a añadir los encargos eliminados uno a uno en el mismo orden en que fueron eliminados, manteniendo la factibilidad de cada solución parcial y de forma que el coste añadido sea mínimo.

Observación 40. Se pretende minimizar el número total de encargos que es necesario eliminar para llegar a una solución parcial factible. Posteriormente, la reinserción de los encargos en el mismo orden en que fueron eliminados da prioridad a los encargos que provocaban una mayor infactibilidad IP y que por ello pueden ser más difíciles de recolocar.

Ejemplo 10. Volvamos al Ejemplo 6 para ilustrar cómo funciona el operador $\breve{\Pi}(S)$. Como ya se determinó en el Ejemplo 7, $B_S^* = \{u_1, u_2\} = \{1, 6\}$ y $B_S^o = (1, 6)$, por lo que $\breve{\Pi}(S) = \dot{\Upsilon}_6 \circ \dot{\Upsilon}_1 \circ \Upsilon_6 \circ \Upsilon_1(S)$. Ya sabemos que $\Upsilon_6 \circ \Upsilon_1(S) = \tilde{S}$, que es una solución $\{1, 6\}$-parcial factible. Así, la solución final resultado del operador proyección optimizada $\breve{\Pi}(S) = \dot{\Upsilon}_6 \circ \dot{\Upsilon}_1(\tilde{S})$ se obtiene simplemente añadiendo de nuevo el encargo 1 (resultado del operador $\dot{\Upsilon}_1$) y luego el 6 (resultado del operador $\dot{\Upsilon}_6$) a $\tilde{S}$, escogiendo para tales inserciones las mejores posiciones posibles.

5.3. Combinación de infactibilidades: soluciones α-potenciales

En las secciones anteriores se han tratado las Infactibilidades por Capacidad (IC) y por Precedencias (IP) de forma independiente, considerando por un lado las soluciones α-factibles y por otro las 0-potenciales. Sin embargo, ambas infactibilidades están estrechamente relacionadas y pueden aparecer al mismo tiempo en lo que denominamos soluciones α-potenciales (definición 21). A continuación se estudiará con detalle cómo relacionar y combinar ambos tipos de infactibilidades.

5.3.1. Transformación de infactibilidades

Las soluciones α-factibles (IC) violan las restricciones de capacidad y las 0-potenciales (IP) las de precedencia, originando dos tipos distintos de infactibilidad. Una cuestión interesante es establecer qué relación hay entre ambas, determinando cuál es más difícil de manejar y si una puede transformarse en la otra sin cambiar el coste de la solución.

Dada una solución S con sólo infactibilidad IC (respectivamente IP), se desea estudiar si existe otra solución $\hat{S}$ con el mismo coste que S y con sólo infactibilidad IP (respectivamente

IC). El coste de la solución viene dado por las rutas de recogida y entrega, y por tanto para mantenerlo constante basta con no modificar dichas rutas. En las proposiciones 52-54 se da respuesta a esta cuestión.

Proposición 52. *Sean* $\alpha \in \mathbb{N}$, $S = (\pi_1, \pi_2, \lambda) \in X_\alpha$ *una solución* α*-factible y* $D_e = \{u \in D \,|\, \text{card}(\lambda^{-1}(\lambda(u))) > Q\}$. *Existe una solución* 0*-potencial* $\hat{S} = (\pi_1, \pi_2, \hat{\lambda}) \in \tilde{X}$ *tal que* $\lambda(u) = \hat{\lambda}(u)\, \forall u \in D \backslash D_e$ *y* $\text{card}\{u \in D_e \,|\, \lambda(u) \neq \hat{\lambda}(u)\} \leq m\alpha$.

Demostración

Cualquier asignación de pilas $\hat{\lambda}$ definida de forma que $\hat{\lambda}(u) = \lambda(u)\ \forall u \in D \backslash D_e$ y $\text{card}\, \lambda^{-1}(u) \leq Q\ \forall u \in D_e$ verifica lo pedido. Las imágenes por $\hat{\lambda}$ de los encargos que no están en D_e se mantienen y los demás pueden asignarse a cualquier pila, siempre y cuando ninguna exceda la capacidad máxima, lo cual siempre se puede conseguir ya que $mQ \leq n$. □

La proposición 52 indica que cualquier solución α-factible se puede transformar en 0-potencial, simplemente reasignando los encargos que exceden el tamaño máximo a otras pilas con espacio libre; de este modo se obtienen soluciones que verificarán con seguridad las restricciones de capacidad, aunque posiblemente no las de precedencia. Dicha proposición permite definir un operador para obtener soluciones potenciales a partir de soluciones α-factibles:

Definición 51. Sea $\alpha \in \mathbb{N}$. Se define el operador $T_{CP} : X_\alpha \longrightarrow \tilde{X}$ que asigna a cada solución $S \in X_\alpha$ una solución potencial $T_{CP}(S)$ con el mismo coste, cuya existencia garantiza la proposición 52.

Proposición 53. *Sea* $S = (\pi_1, \pi_2, \lambda) \in \tilde{X}$ *una solución* 0*-potencial,* $\alpha \in \mathbb{N}$ *y para cada* $u \in D$ *sea* $\text{EC}(u) = \{v \in D \,|\, (\pi_1^{-1}(u) - \pi_1^{-1}(v)) \cdot (\pi_2^{-1}(u) - \pi_2^{-1}(v)) < 0\}$. *Si* $\exists u_1, \cdots, u_m \in D$ *tal que* $\text{EC}(u_i) = \varnothing$, $\forall i = 1, \cdots, m$, *entonces no existe* $\hat{\lambda} \in \text{Apl}(D, P)$ *de manera que* $(\pi_1, \pi_2, \hat{\lambda}) \in X_\alpha$ *sea una solución* α*-factible.*

Demostración

Si $\text{EC}(u) = \varnothing$, las rutas de recogida y entrega de la solución hacen que el encargo u sea incompatible con todos los demás, de manera que en ninguna solución factible podría estar asignado a la misma pila que ningún otro. Por tanto, si existen m encargos con esta propiedad, cualquier solución α-factible requerirá al menos $m + 1$ pilas disponibles, por lo que no existirá ninguna solución α-factible con sólo m pilas. □

Proposición 54. *Sea* $S = (\pi_1, \pi_2, \lambda) \in \tilde{X}$ *una solución* 0*-potencial,* $\alpha \in \mathbb{N}$ *y para cada* $u \in D$ *sea* $\text{EC}(u) = \{v \in D \,|\, (\pi_1^{-1}(u) - \pi_1^{-1}(v)) \cdot (\pi_2^{-1}(u) - \pi_2^{-1}(v)) < 0\}$. *Si* $\exists u_1, \cdots, u_{m+1} \in D$

tal que $u_j \notin \mathrm{EC}(u_i)$, $\forall i, j = 1, \cdots, m+1$, $i \neq j$, *entonces no existe* $\hat{\lambda} \in \mathrm{Apl}(D, P)$ *de manera que* $(\pi_1, \pi_2, \hat{\lambda}) \in X_\alpha$ *sea una solución* α*-factible.*

Demostración

La condición $\exists u_1, \cdots, u_{m+1} \in D$ tal que $u_j \notin EC(u_i)$. $\forall i, j = 1, \cdots, m+1$, $i \neq j$, significa que existen $m+1$ encargos que son incompatibles dos a dos, es decir, ningún par de ellos puede asignarse a la misma pila. Esto obliga a que cualquier solución α-factible utilice al menos $m+1$ pilas, una para cada encargo $u_1, \cdots, u_{m+1}$, por lo que no existirá ninguna solución α-factible con sólo m pilas. □

Las proposiciones 53 y 54 indican que existen soluciones potenciales que no pueden transformarse en α-factibles sólo modificando su asignación de pilas. La condición que se exige en la proposición 53 para asegurar que la transformación de infactibilidad IP en IC no es posible es más fuerte que la pedida en la proposición 54, ya que si existen m encargos incompatibles con todos los demás (que no pueden ser asignados a la misma pila), añadiendo cualquier otro se tienen $m+1$ encargos incompatibles dos a dos.

Existen ejemplos sencillos que ilustran las proposiciones 53 y 54, como los que se presentan a continuación.

Ejemplo 11. Sea una instancia del DTSPMS con $n = 6$, $m = 3$ y $Q = 2$, y sea $S = (\pi_1, \pi_2, \lambda)$ una solución potencial con $\pi_1 = \pi_2 = [1, 2, 3, 4, 5, 6]$. Entonces se tiene que $\mathrm{EC}(u) = \varnothing, \forall u = 1, \cdots, 6$, ya que las rutas de recogida y entrega son exactamente opuestas, y por tanto S no es α-factible para cualquier elección de λ y α.

El Ejemplo 11 sirve para ilustrar tanto la proposición 53 como la 54, ya que la solución S propuesta en dicho ejemplo verifica las condiciones pedidas en ambas proposiciones. Sin embargo esto puede no ser así, ya que existen soluciones que verifican la condición pedida en la proposición 54 pero no la pedida en la 53, como muestra el Ejemplo 12.

Ejemplo 12. Sea una instancia del DTSPMS con $n = 4$, $m = 2$ y $Q = 2$, y sea $S = (\pi_1, \pi_2, \lambda)$ una solución potencial con $\pi_1 = [1, 2, 3, 4]$, $\pi_2 = [4, 1, 2, 3]$. Se tiene que EC(1)=EC(2)=EC(3)=$\{4\}$ y EC(4)=$\{1, 2, 3\}$, por lo que $\mathrm{EC}(u) \neq \varnothing, \forall u = 1, \cdots, 4$ y S no verifica la condición de la proposición 53. Sin embargo el conjunto de encargos $H = \{1, 2, 3\}$ verifica la condición de la proposición 54, $u \notin \mathrm{EC}(v)$, $\forall u, v \in H$, $u \neq v$, que indica que los encargos $\{1, 2, 3\}$ deben asignarse a pilas diferentes. Así, son necesarias al menos 3 pilas para formar una solución factible con las rutas π_1 y π_2, por lo que S no es α-factible para cualquier elección de λ y α.

La proposición 52 establece que cualquier solución α-factible puede transformarse en una solución potencial con las mismas rutas de recogida y entrega, de manera que la infactibilidad por Capacidad se puede transformar fácilmente en infactibilidad por Precedencias.

Por el contrario, las proposiciones 53 y 54 establecen que hay soluciones potenciales que no pueden transformarse en soluciones α-factibles, de manera que la infactibilidad por Precedencias no siempre puede transformarse en infactibilidad por Capacidad. Este hecho sugiere, como se comprueba en la práctica, que la infactibilidad por Precedencias es más difícil de manejar y eliminar que la infactibilidad por Capacidad.

5.3.2. α-Proyección de soluciones α-factibles

El objetivo de introducir soluciones intermedias infactibles en el proceso de búsqueda es permitir la obtención final de buenas soluciones factibles, creadas aplicando métodos adecuados de eliminación de infactibilidades a las soluciones intermedias infactibles. Los métodos presentados previamente modifican la asignación de pilas de la solución para que todas las restricciones de capacidad (algoritmos de reducción de pilas) y precedencia (operadores de proyección) se verifiquen, tratando de modificar lo menos posible las rutas para que el coste de la solución no aumente o lo haga lo menos posible.

Los Algoritmos de Reducción de Pilas presentados en la Sección 5.1.2 permiten transformar algunas soluciones α-factibles en soluciones factibles con el mismo coste, pero no garantizan la obtención de una solución factible en todos los casos, como sí hacen los algoritmos de proyección de soluciones potenciales. Por ello, es necesario diseñar un operador de proyección para construir soluciones factibles a partir de cualquier solución α-factible. El mayor inconveniente de este nuevo operador es que, al garantizar la obtención de una solución proyectada que respeta el tamaño máximo Q de las pilas para toda solución inicial, el coste de la solución proyectada generalmente aumentará.

Según la proposición 52, toda solución α-factible se puede transformar en una solución potencial con las mismas rutas de entrega y recogida; así, una forma sencilla de obtener una solución factible a partir de cualquier solución $S = (\pi_1, \pi_2, \lambda) \in X_\alpha$ sería aplicar alguno de los operadores de proyección para soluciones potenciales a la solución potencial $\hat{S} = (\pi_1, \pi_2, \hat{\lambda})$ dada por la proposición 52.

Proposición 55. *Los operadores de proyección* $\Pi \circ T_{CP}, \breve{\Pi} \circ T_{CP} : X_\alpha \longrightarrow X$ *están bien definidos.*

Demostración

Por la proposición 52 se tiene que $T_{CP}(S) \in \tilde{X}\ \forall S \in X_\alpha$, y por tanto $\Pi(T_{CP}(S))$ y $\breve{\Pi}(T_{CP}(S))$ son soluciones factibles $\forall S \in X_\alpha$. □

El operador $\breve{\Pi}$ se puede adaptar fácilmente para que pueda aplicarse directamente a soluciones α-factibles, eliminando los encargos que excedan el tamaño de las pilas de la solución en lugar de aquellos que violen las restricciones de precedencia.

Proposición 56. *Sea* $S = (\pi_1, \pi_2, \lambda) \in X_\alpha$. *Para cada* $p \in P$ *sea* $DE_p \subset \lambda^{-1}(p)$ *tal que* $\text{card}(DE_p) = \text{máx}\{0, \text{card}\,\lambda^{-1}(p) - Q\}$ *y sean* $\{u_1, \cdots, u_{j^*}\} = DE = \bigcup_{p\in P} DE_p$. *Entonces* $S^* = \Upsilon_{DE}(S)$ *es una solución DE-parcial* factible *y* $\dot{\Upsilon}_{u_{j^*}} \circ \cdots \circ \dot{\Upsilon}_{u_1} \circ \Upsilon_{u_{j^*}} \circ \cdots \circ \Upsilon_{u_1}(S)$ *es una solución factible y completa.*

Demostración

El cardinal de cada conjunto DE_p, $p \in P$, es exactamente el número de encargos que sobrepasan la capacidad máxima Q de la pila p en la solución α-factible S, y por tanto al eliminar todos esos encargos es claro que se obtiene una solución parcial factible, $\Upsilon_{u_{j^*}} \circ \cdots \circ \Upsilon_{u_1}(S)$. Al aplicar secuencialmente sobre dicha solución parcial los operadores $\dot{\Upsilon}_{u_1}, \cdots, \dot{\Upsilon}_{u_{j^*}}$, se añaden de nuevo los encargos previamente eliminados sin aumentar la infactibilidad de la solución, obteniéndose una solución factible y completa. □

Para definir con precisión un operador de proyección para soluciones α-factibles a partir de la proposición 56 es necesario fijar de alguna manera los encargos $\{u_1, \cdots, u_{j^*}\}$ que se van a modificar, ya que hay muchas formas de construir los conjuntos $DE_p \subset \lambda^{-1}(p)$, para los cuales lo único fijo es su cardinal. Siguiendo el criterio utilizado para diseñar los algoritmos de Reducción de Pilas, es razonable elegir los encargos recogidos en las últimas posiciones, ya que éstos son los que ocupan las cabeceras de las pilas y pueden reasignarse de forma más sencilla. A partir de esta idea se obtiene el operador de la definición 52.

Definición 52. El *operador* α*-proyección optimizada* $\breve{\Pi}_\alpha : X_\alpha \longrightarrow X$ se define, para cada solución α-factible $S \in X_\alpha$, de la siguiente forma:

$$\breve{\Pi}_\alpha(S) = \dot{\Upsilon}_{u_{j^*}} \circ \cdots \circ \dot{\Upsilon}_{u_1} \circ \Upsilon_{u_{j^*}} \circ \cdots \circ \Upsilon_{u_1}(S) \tag{5.10}$$

donde $\{u_1, \cdots, u_{j^*}\} = DE_S = \bigcup_{p\in P} DE_p$, $DE_p \subset \lambda^{-1}(p)$, $\text{card}(DE_p) = \text{máx}\{\text{card}\,\lambda^{-1}(p) - Q, 0\}$ tales que para cada $p \in P$ se verifica que $\pi_1^{-1}(u) > \pi_1^{-1}(v)$, $\forall u \in DE_p$, $\forall v \in \lambda^{-1}(p) \backslash DE_p$. El conjunto DE_S, formado por los encargos que exceden el tamaño máximo de las pilas de la solución S, se denomina *conjunto de máxima infactibilidad IC de S.*

El Ejemplo 13 ilustra el funcionamiento del operador $\breve{\Pi}_\alpha$ recientemente introducido en la definición 52.

Ejemplo 13. Consideremos una instancia del DTSPMS con 9 encargos $D = \{1, 2, \cdots, 9\}$, 3 pilas disponibles $P = \{1, 2, 3\}$ y un tamaño máximo por pila de $Q = 3$ unidades, y la solución 2-factible $S = (\pi_1, \pi_2, \lambda)$ definida de la siguiente manera:

$$\pi_1 \equiv \text{ [0-1-2-3-5-6-7-8-4-9-0]} \qquad \pi_2 \equiv \text{ [0-4-9-7-8-6-5-3-2-1-0]}$$

$$\lambda(c) = \begin{cases} 1 & \text{si } c \in \{1,2,3,7,4\} \\ 2 & \text{si } c \in \{5,6\} \\ 3 & \text{si } c \in \{8,9\} \end{cases}$$

La representación esquemática de la solución S se presenta en la Figura 5.12.

π_1: 0-1-2-3-5-6-7-8-4-9-0
π_2: 0-4-9-7-8-6-5-3-2-1-0

Pilas:

4		
7		
3		
2	6	9
1	5	8

Figura 5.12: Representación de la solución S del Ejemplo 13

Como las pilas 2 y 3 ya albergan dos encargos y $Q = 3$, dichas pilas sólo tienen capacidad para albergar un único encargo nuevo, y por tanto la única forma de recolocar de forma factible los encargos situados en las cabeceras de las pilas con exceso, que son el 4 y el 7, sería recolocar uno en la pila 2 y otro en la 3. Sin embargo, el encargo 4 no puede reasignarse a la pila 3, ya que se recoge y se entrega antes que el 9, y por la misma razón el encargo 7 tampoco puede reasignarse a la pila 3, ya que se recoge y entrega antes que el 8. Por tanto es imposible obtener una solución con pilas de tamaño 3 y el mismo coste que S sólo moviendo los encargos 4 y 7, y es claro que cualquiera de los algoritmos de Reducción de Pilas sería incapaz de obtener una solución factible en este caso.

Para estas soluciones en las cuales los algoritmos de Reducción de Pilas fallan es donde el operador de proyección $\breve{\Pi}_\alpha$ es útil. Como hay que recolocar los encargos 4 y 7 se tiene que $\alpha = 2$, $j^* = 2$ y $DE_S = \{4,7\}$. La solución DE_S-parcial $\tilde{S} = \Upsilon_7 \circ \Upsilon_4(S)$ obtenida tras eliminar dichos encargos, que ahora sí verifica las restricciones de capacidad, corresponde a la primera solución representada en la Figura 5.13, donde los dibujos de las pilas hacen referencia a la asignación de pilas de la solución y las secuencias colocadas encima y debajo representan las rutas de recogida y entrega, respectivamente.

El siguiente paso sería aplicar el operador $\dot{\Upsilon}_4$, reinsertando el encargo 4 en la mejor posición posible, y posteriormente aplicar el operador $\dot{\Upsilon}_7$, haciendo lo mismo con el encargo 7. Si el encargo 4 se asigna a la pila 3 no se pueden mantener las posiciones originales en las rutas de recogida y entrega; por lo tanto, el operador $\dot{\Upsilon}_4$ asignará el encargo 4 a la pila 2 y lo colocará entre los encargos 8 y 9 en la ruta de entrega y en primera posición en la ruta de recogida. La solución obtenida, $\dot{\Upsilon}_4(\tilde{S})$, corresponde a la segunda solución representada en la Figura 5.13, donde se observa que se ha obtenido una solución $\{7\}$-parcial factible.

Ahora se aplica $\dot{\Upsilon}_7$ sobre $\dot{\Upsilon}_4(\tilde{S})$, donde la única posibilidad es asignar el encargo 7 a la pila 3; las posiciones originales no se pueden mantener, por lo que habría que buscar aquellas que supongan un incremento menor en la longitud total recorrida respetando las

relaciones de precedencia con los encargos 8 y 9. Dichas posiciones dependen de la matriz de distancias, pero normalmente estarán próximas a las posiciones originales. Supongamos que dichas posiciones colocan el encargo 7 en su posición original en la ruta de recogida, entre el 6 y el 8, y entre el 8 y el 6 en la ruta de entrega.

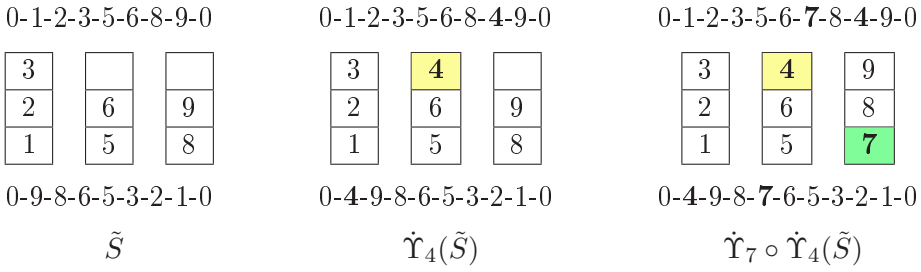


Figura 5.13: Construcción de $\breve{\Pi}_2(S)=\dot{\Upsilon}_7 \circ \dot{\Upsilon}_4 \circ \Upsilon_7 \circ \Upsilon_4(S)$

Así, el resultado del operador 2-proyección optimizada es $\breve{\Pi}_2(S)=\dot{\Upsilon}_7 \circ \dot{\Upsilon}_4 \circ \Upsilon_7 \circ \circ \Upsilon_4(S)$, que corresponde a la tercera solución representada en la Figura 5.13, donde se observa que se trata de una solución completa y factible. Nótese que $\breve{\Pi}_2(S)$ es una solución factible que sólo difiere de S en las posiciones de los encargos 7 y 8 en la ruta de entrega, que han sido intercambiadas.

5.3.3. α-Proyección de soluciones α-potenciales

Los dos tipos de infactibilidad (IC e IP) considerados hasta el momento pueden combinarse y utilizarse conjuntamente en lo que denominamos soluciones α-potenciales. Los algoritmos y operadores introducidos para tratar las infactibilidades se definen para soluciones α-factibles o para soluciones potenciales, pero pueden adaptarse para que consideren ambas infactibilidades y puedan tratar directamente soluciones α-potenciales.

La proyección optimizada definida en las secciones anteriores consiste fundamentalmente en eliminar los encargos que producen la infactibilidad y reinsertarlos secuencialmente hasta la obtención de una solución final factible. Este mecanismo puede aplicarse directamente a soluciones α-potenciales como se detalla en la proposición 57, eliminando los encargos con infactibilidades IC e IP que sea necesario e introduciéndolos posteriormente verificando todas las restricciones de capacidad y precedencia.

Proposición 57. *Sean $S \in \tilde{X}_\alpha$ una solución α-potencial, B^*_S, $B^0_S = (u_1, \cdots, u_{i^*})$ su conjunto y vector de máxima infactibilidad IP, respectivamente, y sea $DE(\Upsilon_{B^*_S}(S)) = DE = \{u_{i^*+1}, \cdots, u_{j^*}\}$ el conjunto de máxima infactibilidad IC de la solución B^*_S-parcial $\Upsilon_{B^*_S}(S)$, construido siguiendo la proposición 52. Entonces $\Upsilon_{B^*_S \cup DE}(S)$ es una solución parcial factible y $\dot{\Upsilon}_{u_{j^*}} \circ \cdots \circ \dot{\Upsilon}_{u_1} \circ \Upsilon_{u_{j^*}} \circ \cdots \circ \Upsilon_{u_1}(S)$ es una solución factible y completa.*

Demostración

La solución parcial $\Upsilon_{B_S^*}(S)$ obtenida eliminando de S los encargos $\{u_1, \cdots, u_{i^*}\}$ de su conjunto de máxima infactibilidad IP es α-factible, ya que S era α-potencial, y por tanto se puede construir su conjunto de máxima infactibilidad IC, que denotamos por $DE = \{u_{i^*+1}, \cdots, u_{j^*}\}$. Así, $\Upsilon_{u_{j^*}} \circ \cdots \circ \Upsilon_{u_{i^*+1}}(\Upsilon_{B_S^*}(S)) = \Upsilon_{u_{j^*}} \circ \cdots \circ \Upsilon_{u_1}(S)$ es una solución $\{u_1, \cdots, u_{j^*}\}$-parcial factible. Por tanto, $\dot{\Upsilon}_{u_{j^*}} \circ \cdots \circ \dot{\Upsilon}_{u_1} \circ \Upsilon_{u_{j^*}} \circ \cdots \circ \Upsilon_{u_1}(S)$ es una solución factible y completa, como se quería demostrar. □

La proposición 57 permite definir el operador de α-proyección optimizada de soluciones α-potenciales, que permitiendo un pequeño abuso de notación denotaremos también por $\breve{\Pi}_\alpha$. De esta manera, el operador $\breve{\Pi}_\alpha$ aplicado a una solución α-factible será el de la definición 52 y aplicado a una solución α-potencial será el de la definición 53.

Definición 53. El *operador α-proyección optimizada* $\breve{\Pi}_\alpha : \tilde{X}_\alpha \longrightarrow X$ se define, para cada solución α-potencial $S \in \tilde{X}_\alpha$, como $\breve{\Pi}_\alpha(S) = \dot{\Upsilon}_{u_{j^*}} \circ \cdots \circ \dot{\Upsilon}_{u_1} \circ \Upsilon_{u_{j^*}} \circ \cdots \circ \Upsilon_{u_1}(S)$, donde $B_S^0 = (u_1, \cdots, u_{i^*})$ y $DE(\Upsilon_{B_S^*}(S)) = DE = \{u_{i^*+1}, \cdots, u_{j^*}\}$.

El Ejemplo 14 ilustra el operador α-proyección optimizada introducido en la definición 53.

Ejemplo 14. Consideremos una instancia del DTSPMS con 9 encargos $D = \{1, 2, \cdots, 9\}$, 3 pilas disponibles $P = \{1, 2, 3\}$ y un tamaño máximo por pila de $Q = 3$ unidades, y la solución 1-potencial $S = (\pi_1, \pi_2, \lambda)$ definida de la siguiente manera:

$$\pi_1 \equiv \text{ [0-1-2-3-4-5-6-7-8-9-0]} \qquad \pi_2 \equiv \text{ [0-9-8-7-6-5-3-1-2-4-0]}$$

$$\lambda(c) = \begin{cases} 1 & \text{si } c \in \{1,2,3,4\} \\ 2 & \text{si } c \in \{5\} \\ 3 & \text{si } c \in \{6,7,8,9\} \end{cases}$$

La representación esquemática de la solución S se presenta en la Figura 5.14.

π_1: 0-1-2-3-4-5-6-7-8-9-0
π_2: 0-9-8-7-6-5-3-1-2-4-0

Pilas:

4		9
3		8
2		7
1	5	6

Figura 5.14: Representación de la solución S del Ejemplo 14

Para aplicar el operador 1-proyección optimizada $\breve{\Pi}_1$ a la solución S lo primero que hay que hacer es construir el vector de máxima infactibilidad IP $B_S^0 = (u_1, \cdots, u_{i^*})$. Se tiene

que $\Lambda_P^1[\varnothing, S](1) = 2$, $\Lambda_P^1[\varnothing, S](2) = 2$, $\Lambda_P^1[\varnothing, S](3) = 1$, $\Lambda_P^1[\varnothing, S](4) = 3$ y $\Lambda_P^1[\varnothing, S](u) = 0$ $\forall u \in \{5, 6, 7, 8, 9\}$, por lo que el primer elemento del vector B_S^0 será el encargo 4, que es el de mayor medida de infactibilidad IP. Así, siguiendo la definición 48, se tiene que $U_1 = \{4\}$ y $S_1 = \Upsilon_{\{4\}}(S)$ es la solución parcial obtenida eliminando el encargo 4, que viene representada en la Figura 5.15.

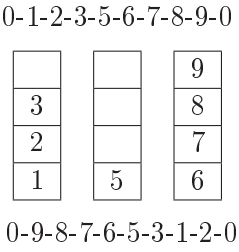


Figura 5.15: Representación de la solución S_1 del Ejemplo 14

Ahora hay que calcular las medidas $\Lambda_P^1[U_1, S_1]$ de los encargos de la solución parcial S_1 para decidir cuál será el siguiente elemento del vector B_S^0. Se tiene que $\Lambda_P^1[U_1, S_1](1) = \Lambda_P^1[U_1, S_1](2) = 2$ y $\Lambda_P^1[U_1, S_1](u) = 0$ $\forall u \in \{3, 5, 6, 7, 8, 9\}$, por lo que hay empate entre los encargos 1 y 2 y elegimos por ejemplo el encargo 1. Así, siguiendo de nuevo la definición 48, se tiene que $U_2 = \{1\}$ y $S_2 = \Upsilon_{\{1\}}(S_1)$ es la solución parcial obtenida eliminando de S_1 el encargo 1, que viene representada en la Figura 5.16.

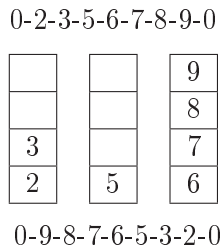


Figura 5.16: Representación de la solución S_2 del Ejemplo 14

Ahora de nuevo habría que calcular las medidas $\Lambda_P^1[U_2, S_2]$ de los encargos de la solución parcial S_2 para decidir cuál será el siguiente elemento del vector B_S^0. Sin embargo, se tiene que $\Lambda_P^1[U_2, S_2](u) = 0$ $\forall u \in \{2, 3, 5, 6, 7, 8, 9\}$, por lo que la solución parcial S_2 es 1-factible y por tanto $B_S^0 = (4, 1)$.

Una vez calculados B_S^0 y $S_2 = \Upsilon_{B_S^*}(S)$ hay que calcular $DE(S_2)$, el conjunto de máxima infactibilidad IC de S_2. El encargo 9 es el único encargo de S_2 que excede el tamaño máximo de la pilas, por lo que $DE(S_2) = \{9\}$.

Así, siguiendo la notación de la definición 53 se tiene que $\alpha = 1$, $i^* = 2$, $j^* = 3$,

$B_S^0 = (4, 1)$, $DE(S_2) = \{9\}$, y como consecuencia $\breve{\Pi}_1(S) = \dot{\Upsilon}_9 \circ \dot{\Upsilon}_1 \circ \dot{\Upsilon}_4 \circ \Upsilon_9 \circ \Upsilon_1 \circ \Upsilon_4(S)$.

La solución parcial $S_3 = \Upsilon_9 \circ \Upsilon_1 \circ \Upsilon_4(S) = \Upsilon_9(S_2)$, correspondiente a la primera representación de la Figura 5.17, es factible, de manera que la solución final factible y completa $\breve{\Pi}_1(S) = \dot{\Upsilon}_9 \circ \dot{\Upsilon}_1 \circ \dot{\Upsilon}_4(S_3)$ se obtiene reinsertando los encargos eliminados 4, 1 y 9 en S_3 en las mejores posiciones posibles. La segunda representación de la Figura 5.17 corresponde a una de las soluciones $\breve{\Pi}_1(S)$ posibles (los mejores lugares de inserción en las rutas dependen de la matriz de costes).

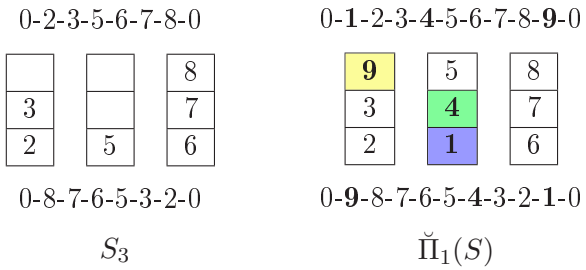


Figura 5.17: Representaciones de las soluciones S_3 y $\breve{\Pi}_1(S)$ del Ejemplo 14

5.3.4. Función objetivo para la búsqueda exterior

En la mayor parte de los algoritmos heurísticos las soluciones por las que se mueve el proceso de búsqueda son factibles, y el objetivo es siempre minimizar el coste total de la solución. Sin embargo, cuando se trata de algoritmos que permiten que las soluciones intermedias puedan violar algunas restricciones del problema, los cuales se denominarán *algoritmos exteriores* (ya que se mueven por fuera del espacio de soluciones factibles), estas soluciones pueden estar lejos de la factibilidad, de manera que si el único objetivo que guiase el proceso de búsqueda fuera minimizar el coste total de la solución sería complicado recuperar la factibilidad inicial y obtener soluciones finales factibles.

Por esa razón la función objetivo utilizada habitualmente en los algoritmos exteriores para guiar el proceso de búsqueda tiene varios sumandos, uno siempre asociado al coste de la solución y los demás a sus medidas de infactibilidad. Todos los sumandos (*objetivos*) deben ser minimizados para obtener soluciones factibles de buena calidad, pero pueden tener mayor o menor importancia en distintas fases del algoritmo (en un principio la infactibilidad se puede sacrificar en pos de disminuir el coste, pero en última instancia lo que se requiere es encontrar soluciones factibles). Para determinar la importancia de cada objetivo, los sumandos de la función objetivo se ponderan con unos pesos que van cambiando dinámicamente a lo largo de la ejecución del algoritmo.

Para que la ponderación de pesos sea correcta es necesario *normalizar* las medidas de

infactibilidad y coste de la solución, de manera que las unidades utilizadas sean comparables. Para ello, para cada instancia concreta que se vaya a resolver se calcularán el coste medio y las infactibilidades medias de una solución, siguiendo las expresiones dadas en la definición 54.

Definición 54. Dada una instancia del DTSPMS, el coste medio, denotado por $\bar{C}$, y las infactibilidades medias, denotadas por $\bar{\Lambda}_C^i$ (infactibilidades IC) y $\bar{\Lambda}_P^i$ (infactibilidades IP), $i = 1, 2, 3$, de dicha instancia se definen en (5.11), (5.12) y (5.13), respectivamente.

$$\bar{C} = \frac{1}{n+1} \sum_{\substack{i,j \in V^1 \\ i \neq j}} c_{ij}^1 + \frac{1}{n+1} \sum_{\substack{i,j \in V^2 \\ i \neq j}} c_{ij}^2 \tag{5.11}$$

$$\Lambda_C^1 = \frac{m}{2} \cdot \frac{1}{2} = \frac{m}{4}, \qquad \Lambda_C^2 = \frac{\alpha m}{2} \cdot \frac{1}{2} = \frac{\alpha m}{4}, \qquad \Lambda_C^3 = \frac{\alpha}{2} \tag{5.12}$$

$$\Lambda_P^1 = \frac{m}{2} \cdot \frac{Q(Q-1)}{2} = \frac{mQ(Q-1)}{4}, \qquad \Lambda_P^2 = \frac{m}{2}, \qquad \Lambda_P^3 = \frac{n}{2} \tag{5.13}$$

Así, las medidas de infactibilidad y coste de una solución se normalizan simplemente dividiéndolas por $\bar{\Lambda}_C^i$, $\bar{\Lambda}_P^i$ y $\bar{C}$, respectivamente. Fijando los pesos asociados a cada sumando normalizado definimos una nueva función objetivo en la que se ponderan el coste y la infactibilidad de la solución:

Definición 55. Sean $\eta_C^i, \eta_P^i \in [0, 1]$, $i = 1, 2, 3$, los pesos asociados a las medidas de infactibilidad IC e IP, respectivamente, y $\eta_C = 1 - \sum_{i=1}^3 (\eta_C^i + \eta_P^i)$ el peso asociado al coste de la solución. Denotemos también por $\eta = (\eta_C^1, \eta_C^2, \eta_C^3, \eta_P^1, \eta_P^2, \eta_P^3)$ el vector ordenado de pesos de las infactibilidades. La *función objetivo η-ponderada* se denota por z_η y se define como:

$$z_\eta(S) = \frac{\eta}{\bar{C}} z(S) + \sum_{i=1}^3 \frac{\eta_C^i}{\bar{\Lambda}_C^i} \Lambda_C^i(S) + \sum_{i=1}^3 \frac{\eta_P^i}{\bar{\Lambda}_P^i} \Lambda_P^i(S), \quad S \in \tilde{X}_\alpha$$

Además de mejorar el coste o la infactibilidad de una solución intermedia, puede resultar interesante en cierto momento dar prioridad al balanceo del tamaño de las pilas, ya que algunos algoritmos permiten la utilización de algunas unidades de tamaño extra. Esto da lugar a otra función objetivo que mide la diferencia entre el tamaño máximo de las pilas de la solución inicial y el tamaño máximo de las pilas de la solución evaluada. Cuanto mayor sea esta diferencia más se habrá conseguido balancear el tamaño de las pilas, y por tanto el objetivo será maximizar esta función.

Definición 56. Sea $S = (\pi_1, \pi_2, \lambda) \in \tilde{X}_\alpha$ una solución α-potencial. La *función objetivo S-balanceada*, que mide la diferencia del tamaño máximo de las pilas de las soluciones

$\bar{S} \in \tilde{X}_\alpha$ con respecto a la solución S, se denota por z_S y se define de la siguiente forma:

$$z_S(\bar{S}) = z_S(\bar{\pi}_1, \bar{\pi}_2, \bar{\lambda}) = \max_{k \in P}\{|\lambda^{-1}(k)|\} - \max_{k \in P}\{|\bar{\lambda}^{-1}(k)|\}, \quad \bar{S} = (\bar{\pi}_1, \bar{\pi}_2, \bar{\lambda}) \in \tilde{X}_\alpha$$

Los algoritmos, operadores y medidas de infactibilidad introducidos en este capítulo, junto con las estrategias de generación de soluciones iniciales y las estructuras de entornos introducidas en el Capítulo 4, se utilizarán en el capítulo siguiente para el diseño de algoritmos heurísticos para el DTSPMS.

Capítulo 6

Resolución del DTSPMS mediante heurísticas

En este capítulo se describen con detalle varios algoritmos heurísticos para resolver el DTSPMS que utilizan las técnicas introducidas en los Capítulos 4 y 5. En la Sección 6.1 se proponen varios algoritmos de Búsqueda en Entorno Variable basados en la utilización de los entornos presentados en la Sección 4.3 y en la inclusión de algunas técnicas propias de otras heurísticas, además de la introducción de soluciones α-factibles. En la Sección 6.2 se adapta un algoritmo de Temple Simulado al DTSPMS utilizando para la búsqueda local el entorno para el que se ha observado un mejor comportamiento global de los descritos en las Secciones 4.3.1-4.3.7. En la Sección 6.3 se introduce un algoritmo, denominado Algoritmo Exterior, que utiliza soluciones intermedias no factibles (α-potenciales) para flexibilizar el proceso de búsqueda; las principales herramientas que utilizará dicho algoritmo fueron introducidas en el Capítulo 5, donde también se estudió con detalle cómo manejar los distintos tipos de infactibilidad considerados.

Todos los algoritmos propuestos en las Secciones 6.1-6.3 toman como parámetros de entrada una solución inicial y un tiempo máximo de cómputo, e intentan encontrar la mejor solución posible a partir de dicha solución inicial sin exceder el máximo tiempo de cómputo concedido. Sin embargo, dichos algoritmos pueden ejecutarse con distintas soluciones iniciales e incluso combinarse entre sí para conseguir soluciones finales de mayor calidad. En la Sección 6.4 se presenta un algoritmo genérico de Multiarranque con Intensificación que sirve de estructura para utilizar y/o combinar los algoritmos heurísticos presentados en las secciones anteriores y ejecutarlos con distintas soluciones iniciales.

6.1. Búsqueda en Entorno Variable

En esta sección se proponen varios algoritmos heurísticos para el DTSPMS basados en las ideas de la Búsqueda en Entorno Variable. Primero se adaptan los algoritmos estándar de Búsqueda en Entorno Variable (VND, BVNS y GVNS) al DTSPMS, y posteriormente se explica cómo se les pueden añadir elementos de otras heurísticas (técnicas de multiarranque, GRASP, búsqueda tabú) para mejorar su comportamiento al ser aplicados al problema. Finalmente se propone un algoritmo nuevo diseñado específicamente para el DTSPMS que incluye todas las ideas anteriores y algunas más y que se denomina *Búsqueda en Entorno Variable Híbrida*.

6.1.1. Búsqueda Descendente

A continuación se presenta el pseudocódigo de un algoritmo de **Búsqueda en Entorno Variable Descendente** (*Variable Neighborhood Descent*, VND) adaptado al DTSPMS.

Algoritmo 20. Búsqueda en Entorno Variable Descendente (VND)

Parámetros de entrada

- S: Solución inicial factible.
- α: Tamaño adicional permitido en las pilas.
- $\{\Delta_k,\ k = 1, \cdots, n_e\}$: Conjunto finito de estructuras de entornos sobre X contenido en {RS, CS, ISS, R, r-RP, r-SP, r-CSP}.

Parámetros de salida

- S^*: Mejor solución encontrada.

Otros algoritmos utilizados

- RTV: un algoritmo de Reducción de Tamaño Variable.

Pseudocódigo

1. *Inicialización:* Poner $k = 1$, $Q = Q + \alpha$ y $S^* = S$.
2. *Inicio Búsqueda Local:* Poner $mejora = F$.
3. *Búsqueda Local en Δ_k:* Encontrar la mejor solución $\hat{S} \in \Delta_k(S)$ del k-ésimo entorno de S.
4. *Reducción Tamaño Máximo:* Obtener $(\hat{\alpha}, \hat{S})$ =RTV$(\hat{S}, \alpha, 0)$. Si $\hat{\alpha} = 0$ y $z(\hat{S}) < z(S^*)$, poner $S^* = \hat{S}$.

5. Si $z(\hat{S}) < z(S)$, poner $S = \hat{S}$, $mejora = T$ y volver al paso 3.
6. *Cambio de entorno:* Si $mejora$ poner $k = 1$. En otro caso, poner $k = k + 1$.
7. *Criterio de parada:*
   - Si $k \leq n_e$ volver al paso 2.
   - Si $k > n_e$ FIN: la mejor solución encontrada es S^*.

6.1.2. Búsqueda en Entorno Variable Básica y General

A continuación se presenta el pseudocódigo de un algoritmo de **Búsqueda en Entorno Variable Básica** (*Basic Variable Neighborhood Search*, BVNS) adaptado al DTSPMS. Es necesario determinar el procedimiento de Búsqueda Local a utilizar, y se permite la utilización de tamaño adicional en las pilas.

Algoritmo 21. Búsqueda en Entorno Variable Básica (BVNS)

Parámetros de entrada

- S: Solución inicial factible.
- t^*: Tiempo máximo de computación disponible.
- α: Tamaño adicional permitido en las pilas.
- $\{\Delta_k,\ k = 1, \cdots, n_e\}$: Conjunto finito de estructuras de entornos sobre X contenido en {RS, CS, ISS, R, r-RP, r-SP, r-CSP}.
- *reinicio*: Constante booleana que indica si se reinicia el recorrido de los entornos al conseguir mejorar la solución (T) o no (F).
- *shake*: Número de agitaciones que se realizan en cada entorno.
- BL: Procedimiento de Búsqueda Local a utilizar.

Parámetros de salida

- S^*: Mejor solución encontrada.
- S_u: Última solución considerada.

Otros algoritmos utilizados

- RTV: un algoritmo de Reducción de Tamaño Variable.

Pseudocódigo

1. *Inicialización:* Poner $S^* = S$ y $Q = Q + \alpha$.
2. *Inicio Recorrido Entornos:* Poner $k = 1$ y $mejora = F$.
3. *Perturbación en* Δ_k*:*
   3.1 Poner $j = 0$, $\bar{S} = S$.
   3.2 Elegir aleatoriamente una solución $\hat{S}$ del entorno $\Delta_k(\bar{S})$.
   3.3 Poner $j = j + 1$, $\bar{S} = \hat{S}$.
   3.4 Si $j < shake$ volver al paso 3.2.
4. *Búsqueda Local:* Tomando $\hat{S}$ como solución inicial, utilizar el procedimiento de búsqueda local BL para obtener un mínimo local $\bar{S}$.
5. *Reducción Tamaño Máximo:* Obtener $(\hat{\alpha}, \hat{S}) = \text{RTV}(\hat{S}, \alpha, 0)$. Si $\hat{\alpha} = 0$ y $z(\hat{S}) < z(S^*)$, poner $S^* = \hat{S}$.
6. Si $z(\bar{S}) < z(S)$, poner $S = \bar{S}$, $mejora = T$.
7. *Cambio de entorno:* Si $mejora$ y $reinicio$, volver al paso 2. En otro caso, poner $k = k + 1$.
8. *Fin Recorrido Entornos:*
   - Si se ha superado el máximo tiempo de cómputo disponible t^*, FIN. La mejor solución encontrada es S^* y la última solución considerada es $S_u = S$.
   - Si $k \leq n_e$ volver al paso 3.
   - Si $k > n_e$ volver al paso 2.

Un algoritmo de **Búsqueda en Entorno Variable General** (*General Variable Neighborhood Search*, GVNS) es simplemente un algoritmo de BVNS en el que el procedimiento de búsqueda local BL utilizado es una VND. A continuación se presenta el pseudocódigo correspondiente a este algoritmo.

Algoritmo 22. Búsqueda en Entorno Variable General (GVNS)

Parámetros de entrada

- S: Solución inicial factible.
- t^*: Tiempo máximo de computación disponible.
- α: Tamaño adicional permitido en las pilas.
- $\Delta = \{\Delta_k,\ k = 1, \cdots, n_e\}$: Conjunto finito de estructuras de entornos sobre X contenido en {RS, CS, ISS, R, r-RP, r-SP, r-CSP}.

- $\Omega = \{\Omega_k,\ k = 1, \cdots, e_2\}$: Conjunto finito de estructuras de entornos sobre X contenido en {RS, CS, ISS, R, r-RP, r-SP, r-CSP} para la realización del descenso (VND).

- *reinicio*: Constante booleana que indica si se reinicia el recorrido de los entornos al conseguir mejorar la solución (T) o no (F).

- *shake*: Número de agitaciones que se realizan en cada entorno.

Parámetros de salida

- S^*: Mejor solución encontrada.

- S_u: Última solución considerada.

Otros algoritmos utilizados

- RTV: un algoritmo de Reducción de Tamaño Variable.

- BVNS.

- VND.

Pseudocódigo

1. *Definir BL:* Definir el proceso de búsqueda local BL como un Búsqueda Descendente de manera que BL(S)=VND(S,α,Ω).

2. *Búsqueda General:* $(S^*, S_u) = \text{BVNS}\,(S, t^*, \alpha, \Delta, reinicio, shake, \text{BL})$.

6.1.3. Híbridación de la VNS con otras heurísticas

La idea fundamental de la VNS, consistente en utilizar distintas estructuras de entornos y cambiar entre ellas para guiar la búsqueda, proporciona una herramienta simple y al mismo tiempo potente para la resolución de problemas de optimización. Su simplicidad hace que se pueda incorporar fácilmente a los mecanismos de otras metaheurísticas, aumentando en muchos casos la eficacia del algoritmo considerado.

Al incorporar el cambio sistemático de entorno a otras metaheurísticas se obtienen diversas metaheurísticas híbridas. A continuación se presentan algunas de ellas.

6.1.3.1. Búsqueda Multiarranque

La **Búsqueda Multiarranque** (MS, *MultiStart Search*) es una heurística sencilla en la que, cuando la búsqueda se estanca en un mínimo local, simplemente se reinicia la búsqueda desde otra solución inicial distinta. Una forma obvia de introducir este mecanismo en un algoritmo de Búsqueda en Entorno Variable consiste en reiniciar la VNS con otra solución inicial generada aleatoriamente cada vez que la búsqueda obtiene un mínimo local con respecto a todas las estructuras de entornos consideradas. En Belacel et al. (2002) se propone una heurística híbrida de este tipo.

A continuación se presenta un breve pseudocódigo de una heurística híbrida, denominada **Búsqueda en Entorno Variable con Multiarranque** (VNSMS, *Variable Neighborhood Search with MultiStart*), entre un algoritmo de VNS y una MS para el DTSPMS.

Algoritmo 23. Búsqueda en Entorno Variable con Multiarranque (VNSMS)

Parámetros de entrada

- VNS*: Algoritmo de búsqueda en entorno variable que se va a utilizar.
- CP1: Criterio de parada para el algoritmo VNS*.
- CP2: Criterio de parada final para el algoritmo de VNSMS.

Parámetros de salida

- S^*: Mejor solución encontrada.

Otros algoritmos utilizados

- VNS*.

Pseudocódigo

1. *Inicialización:* Generar una solución inicial factible S del DTSPMS. Poner $S^* = S$. Ir al paso 3.
2. *Nueva solución inicial:* Generar una nueva solución inicial S del DTSPMS.
3. *Búsqueda en Entorno Variable:* Llamar a VNS* con solución inicial S y criterio de parada CP1. La solución obtenida es $\hat{S}$.
4. *Actualización mejor solución conocida:* Si $z(\hat{S}) < z(S^*)$ poner $S^* = \hat{S}$.
5. *Condición de parada:*
   - Si no se cumple la condición de parada CP2, volver al paso 2.

- Si se cumple la condición de parada CP2, el algoritmo termina. La mejor solución obtenida es S^*.

El algoritmo de VNS utilizado en la heurística híbrida (denotado por VNS*) puede ser una VNS básica, general, sesgada, o cualquier otra variante, incluso una simple VND. La condición de parada CP1 indica cuándo el algoritmo VNS se estanca y se reinicia la búsqueda desde otra solución inicial. La condición de parada CP2 indica cuándo finaliza el algoritmo de búsqueda con multiarranque, ya sea por alcanzar el número máximo de iteraciones, terminar el tiempo de cómputo disponible, etc. La solución S^* representa la mejor solución encontrada hasta el momento: comienza siendo igual a S y se actualiza después de cada búsqueda local.

Las soluciones iniciales utilizadas en el paso 2 para realizar el multiarranque se pueden generar siguiendo los procedimientos propuestos en la Sección 4.2.

6.1.3.2. GRASP

La parte característica del GRASP se encuentra en la fase constructiva, por lo que una forma natural de hibridizarlo con la Búsqueda en Entorno Variable es utilizar una VNS como búsqueda local de la fase 2. Esta heurística híbrida entre el GRASP y la VNS ha sido utilizada con éxito en Andreatta y Ribeiro (2002), Canuto et al. (2001), Festa et al. (2001), Martins et al. (2000), Ochi et al. (2001) y Ribeiro et al. (2002). Sin embargo en las pruebas computacionales realizadas se ha comprobado que la implementación realizada de esta heurística no proporciona resultados competitivos en el DTSPMS.

6.1.3.3. Búsqueda Tabú

Hay dos formas de hibridizar la Búsqueda Tabú con la Búsqueda en Entorno Variable que surgen de modo natural:

a) Introducir alguna estructura de memoria de las utilizadas en la TS para guiar la búsqueda dentro de la VNS. Este tipo de híbridos se utilizan en Bräysy (2003), Burke et al. (2001), Kovačević et al. (1999) y Rodríguez et al. (2003).

b) Utilizar varias estructuras de entornos diferentes dentro de la propia Búsqueda Tabú y realizar cambios sistemáticos entre ellas para diversificar la búsqueda. En Brimberg et al. (2000) y Davidović et al. (2001) se pueden encontrar aplicaciones de este segundo tipo de híbridos entre TS y VNS.

En este trabajo se da mayor importancia al algoritmo de Búsqueda en Entorno Variable, que es considerado como algoritmo de partida, y por ello se ha elegido la primera de

las citadas estrategias de hibridación. Así, elegido un algoritmo VNS, éste se hibridiza añadiendo una estructura de memoria que almacena los movimientos realizados en cada entorno para evitar repetirlos poco después. De esta manera existe una lista asociada a cada estructura de entornos, en la cual se almacenan las características distintivas de cada movimiento realizado para prohibir su utilización durante las iteraciones siguientes.

Las ideas de la Búsqueda Tabú se pueden introducir en un algoritmo de Búsqueda en Entorno Variable Básica o General en un segundo nivel. La solución elegida aleatoriamente en cada entorno a partir de la que se realiza la búsqueda local se obtiene realizando un movimiento m que permite diversificar el proceso de búsqueda, pero puede empeorar la función objetivo, por lo que es probable que durante la búsqueda local se intente revertir dicho movimiento m. Al deshacer el movimiento se puede volver al anterior mínimo local, por lo que parece conveniente forzar a que el movimiento m no pueda revertirse. Para ello se incluye en la lista tabú de la estructura de entornos correspondiente el movimiento inverso de m, manteniéndolo durante cierto número de iteraciones para evitar la vuelta al anterior mínimo local.

6.1.4. Algoritmo de Búsqueda en Entorno Variable Híbrida

A continuación se presenta el pseudocódigo detallado del algoritmo denominado **Búsqueda en Entorno Variable Híbrida** (HVNS, *Hibridized Variable Neighborhood Search*) diseñado específicamente para el DTSPMS. Este algoritmo está basado en una GVNS a la que se añaden algunas ideas procedentes de otras metaheurísticas (Multiarranque, Búsqueda Tabú, etc.) y algunos elementos nuevos que permiten que la heurística se adapte mejor al DTSPMS, mejorando su eficiencia.

Algoritmo 24. Búsqueda en Entorno Variable Híbrida (HVNS)

Parámetros de entrada

- S: Solución inicial factible.
- t^*: Tiempo máximo de computación disponible.
- α: Tamaño adicional permitido en las pilas.
- $\Delta = \{\Delta_k,\ k = 1, \cdots, e_1\}$: Conjunto finito de estructuras de entornos sobre X contenido en {RS, CS, ISS, R, r-RP, r-CSP} para perturbar la solución.
- $\Omega = \{\Omega_k,\ k = 1, \cdots, e_2\}$: Conjunto finito de estructuras de entornos sobre X contenido en {RS, CS, ISS, R, r-RP, r-CSP} para la realización del descenso (VND).
- *reinicio*: Constante booleana que indica si se reinicia el recorrido de los entornos al conseguir mejorar la solución (T) o no (F).

- *shake*: Número de agitaciones que se realizan dentro del mismo entorno al realizar una perturbación de la solución.
- θ: Número de ordenaciones distintas de $\{\Omega_k\}$ que se consideran para la realización del descenso.
- $R = \{R_1, \cdots, R_\theta\}$: Conjunto de θ permutaciones de $\{1, \cdots, e_2\}$.
- $P = \{P_1, \cdots, P_\theta\}$: Probabilidades con las que se eligen los distintos órdenes $\{R_1, \cdots, R_\theta\}$.
- *itmax*: Número máximo de iteraciones permitidas sin mejorar la solución actual. Cuando se alcanza *itmax* la búsqueda se reinicia desde la mejor solución conocida después de aplicarle una perturbación.
- *pcost*: Porcentaje máximo de desvío en coste con respecto a la mejor solución conocida. Si se sobrepasa la búsqueda se reinicia desde la mejor solución conocida después de aplicarle una perturbación.
- l_{tabu}: Longitudes de las listas tabú que se utilizan para cada operador.

Parámetros de salida

- S^*: Mejor solución encontrada.

Otros algoritmos utilizados

- RTV: un algoritmo de Reducción de Tamaño Variable.
- VND.
- GVNS.

Pseudocódigo

1. *Mejora inicial:* Realizar una búsqueda local en cada entorno de Ω, sin hacer ninguna perturbación. Actualizar S y la mejor solución conocida hasta el momento S^*.
2. *Orden:* Realizar un sorteo para elegir una permutación $R_j \in R$ según las probabilidades dadas en P. Los entornos de Ω se recorrerán al realizar el descenso siguiendo el orden dado por R_j. Sea Ω_j el conjunto de entornos de Ω ordenados según R_j.
3. *Búsqueda General:* $(S^{**}, S) = \text{GVNS}\,(S, 1, \alpha, \Delta, reinicio, shake, \text{VND}(\sqcup, \alpha, \Omega_j))$.
4. Si $z(S^{**}) < z(S^*)$ poner $it_m = 0$ y $S^* = S^{**}$. En otro caso poner $it_m = it_m + 1$.
5. Si $it_m > itmax$ ó $z(S) > z(S^*)(\frac{100+pcost}{100})$, poner $it_m = 0$, $S = S^*$ y perturbar S con los operadores dados en Δ.
6. *Criterio de parada:* Si el tiempo de cómputo transcurrido no supera t^*, volver al paso 2. En otro caso: FIN. La mejor solución encontrada es S^*.

Observación 41. En este algoritmo se incluye una lista tabú para cada operador, con el fin de almacenar los últimos movimientos realizados para no repetirlos próximamente.

6.2. Temple Simulado

Para aplicar el Temple Simulado al DTSPMS es necesario diseñar un proceso de búsqueda local adaptado a dicho problema, para lo cual basta elegir una de las estructuras de entornos para soluciones factibles previamente introducidas. Dicha estructura de entornos debe permitir la exploración de todo el espacio de soluciones factibles, modificando tanto las rutas como la asignación de pilas de las soluciones iniciales. Por este motivo se ha elegido el entorno de Reinserción para la realización de la búsqueda local en el algoritmo de Temple Simulado propuesto.

A continuación se presenta el pseudocódigo del algoritmo de **Temple Simulado con Búsqueda Local mediante Reinserción** (SAR, *Simulated Annealing with Reinsertion*) para el DTSPMS.

Algoritmo 25. Temple Simulado con Búsqueda Local mediante Reinserción (SAR)

Parámetros de entrada

- S: Solución inicial.
- t^*: Tiempo máximo de computación disponible.
- α: Tamaño adicional permitido en las pilas.
- T_i, T_f: Temperaturas inicial y final, respectivamente.
- ρ: Razón de decremento de la temperatura.

Parámetros de salida

- S^*: Mejor solución encontrada.

Otros algoritmos utilizados

- RTV: un algoritmo de Reducción de Tamaño Variable.
- BVNS.
- VND.

Pseudocódigo

1. *Inicialización:* Poner $S^* = S$, $T = T_i$, $nFases = \lceil \frac{\log(T_f/T_i)}{\log \rho} \rceil$, $f = 0$ y $Q = Q + \alpha$.
2. *Inicio nueva fase:* Poner $f = f + 1$.

3. *Búsqueda Local:* Realizar una búsqueda local con solución inicial S utilizando el entorno de Reinserción. Actualizar S con el óptimo local encontrado.
4. *Reducción Tamaño Máximo:* Obtener $(\hat{\alpha}, S) = \text{RTV}(S, \alpha, 0)$. Si $\hat{\alpha} = 0$ y $z(S) < z(S^*)$, poner $S^* = S$.
5. *Criterio de parada fase actual:* Si el tiempo de cómputo transcurrido durante la etapa actual f alcanza el máximo permitido $\frac{t^*}{nFases}$ ir al paso 7.
6. Elegir aleatoriamente una solución $\hat{S} \in \text{R}(S)$ y un número aleatorio $u \in (0, 1)$.
   - Si $\exp(\frac{z(S)-z(\hat{S})}{T}) \geq u$ poner $S = \hat{S}$ y volver al paso 4.
   - En otro caso, volver al paso 5.
7. *Criterio de parada:*
   - Si $f < nFases$ poner $T = \rho T$ y volver al paso 2.
   - En otro caso, FIN: la mejor solución encontrada es S^*.

6.3. Búsqueda Exterior

En esta sección se propone un algoritmo heurístico para el DTSPMS basado en la realización de una búsqueda local sobre el espacio de soluciones α-potenciales del DTSPMS. Lo denominamos *Algoritmo Exterior* (EXT) porque, al permitir el uso de soluciones α-potenciales, el proceso de búsqueda puede moverse por el *exterior* del conjunto factible. Para guiar dicho proceso de búsqueda se utiliza la función objetivo η-ponderada de la definición 55, en la que se tienen en cuenta el coste de las soluciones obtenidas y las medidas de infactibilidad introducidas en el Capítulo 5, y la función objetivo S-balanceada de la definición 56.

A continuación se presenta un pseudocódigo del citado Algoritmo Exterior.

Algoritmo 26. Algoritmo Exterior (EXT)

Parámetros de entrada

- $S = (\pi_1, \pi_2, \lambda)$: Solución inicial (factible o α-potencial).
- t^*: Tiempo de cómputo disponible.
- α: Tamaño adicional permitido en las pilas.
- $\eta = (\eta_C^1, \eta_C^2, \eta_C^3, \eta_P^1, \eta_P^2, \eta_P^3)$: Pesos iniciales (entre 0 y 1) de las medidas de infactibilidad.
- AP: Algoritmo para la Actualización de Pesos.

Parámetros de salida

- S^*: Mejor solución factible encontrada.

Otros algoritmos utilizados

- RTV: un algoritmo de Reducción de Tamaño Variable.

Pseudocódigo

1. *Inicialización:* Poner $\eta_C = 1 - \sum_{i=1}^{3}(\eta_C^i + \eta_P^i)$, $\alpha_0 = \alpha$ y $z^* = \infty$.
2. *Recorrido Entornos:*
   2.1 Partiendo de S, encontrar un mínimo local $\hat{S}$ para la función objetivo η-ponderada z_η según la estructura de entornos de Reinserción en Pila. Poner $S = \hat{S}$.
   2.2 Encontrar la solución $\hat{S} \in \mathrm{RR}_1(S)$ perteneciente al entorno de Reinserción en Ruta 1 que minimice la función objetivo η-ponderada z_η. Poner $S = \hat{S}$.
   2.3 Encontrar la solución $\hat{S} \in \mathrm{RR}_2(S)$ perteneciente al entorno de Reinserción en Ruta 2 que minimice la función objetivo η-ponderada z_η. Poner $S = \hat{S}$.
   2.4 Encontrar la solución $\hat{S} \in \mathrm{RP}(S)$ perteneciente al entorno de Reinserción en Pila que maximice la función objetivo S-balanceada z_S. Poner $S = \hat{S}$.
3. *Reducción de pilas:* Si $\Lambda_P^1(S) = \Lambda_P^2(S) = \Lambda_P^3(S) = 0$:
   3.1 Hacer $(\alpha, S) = \mathrm{RTV}(S, \alpha, 0)$.
   3.2 Si $\alpha = 0$ y $z(S) < z^*$, poner $S^* = S$ y $z^* = z(S)$.
   3.3 Si $\alpha > 0$ y $z(\breve{\Pi}_\alpha(S)) < z^*$, poner $S^* = \breve{\Pi}_\alpha(S)$ y $z^* = z(\breve{\Pi}_\alpha(S))$.
   3.4 Ir al paso 5.
4. *Proyección completa:* Si $z(\breve{\Pi}_\alpha(S)) < z^*$, poner $S^* = \breve{\Pi}_\alpha(S)$ y $z^* = z(\breve{\Pi}_\alpha(S))$.
5. *Criterio de parada:* Si se alcanza el tiempo máximo de cómputo t^*: FIN. Si $z^* < \infty$, la mejor solución encontrada es S^*, en otro caso no se ha encontrado ninguna solución factible.
6. *Actualización Pesos:* Hacer (η,*pert*)=AP(η) y poner $\eta_C = 1 - \sum_{i=1}^{3}(\eta_C^i + \eta_P^i)$.
7. *Perturbación:* Si *pert*=T, perturbar la solución S realizando algunas operaciones de reinserción aleatoriamente.
8. *Iteración:* Volver al paso 2.

El algoritmo parte de una configuración inicial de pesos η y en base a ella se realiza una búsqueda local guiada por la función objetivo η-ponderada en los entornos de Reinserción en Ruta y Reinserción en Pila (pasos 2.1-2.3), obteniéndose una nueva solución S. Después, para balancear los tamaños de las pilas, se realiza una búsqueda local guiada por la función S-balanceada en el entorno de Reinserción en Pila (paso 2.4). Si la solución obtenida tras este proceso verifica las restricciones de precedencia (paso 3) se aplican los algoritmos de reducción de pilas y si es necesario el operador de proyección optimizada para eliminar la infactibilidad por capacidad y obtener una solución factible. Si por el contrario la solución obtenida no verifica algunas restricciones de precedencia (paso 4) se aplica directamente el operador de proyección optimizada para obtener una solución factible, y en cualquiera de los dos casos se actualiza el valor de la mejor solución factible conocida S^*. Si no se cumple el criterio de parada (paso 5) se actualizan los pesos (paso 6), se perturba la solución actual si es necesario (paso 7) y se repite el proceso anterior.

Uno de los parámetros de entrada del algoritmo EXT es lo que hemos denominado algoritmo de Actualización de Pesos (AP), que determina cómo modificar los pesos de la iteración actual para obtener los de la iteración siguiente. En este tipo de situaciones la actualización de los pesos se realiza habitualmente de forma dinámica, incrementando o disminuyendo cada uno en función del incremento o disminución de la medida de infactibilidad correspondiente durante la última iteración. Sin embargo, tras las primeras pruebas realizadas constatamos que esta estrategia no ofrecía buenos resultados para el DTSPMS, ya que mantenía las soluciones demasiado cerca de la factibilidad y restringía en exceso el proceso de búsqueda. En su lugar introdujimos una estrategia lineal con reinicio en la que inicialmente se da un valor pequeño a los pesos asociados a algunas infactibilidades y éstos se van incrementando de forma constante en cada iteración con el fin de ir aumentando la factibilidad de las soluciones obtenidas; cuando se consigue una solución factible ésta se perturba realizando algunas operaciones de reinserción aleatoriamente y se repite el proceso anterior, inicializando los pesos con nuevos valores.

Combinando el algoritmo de Búsqueda en Entorno Variable Híbrida (HVNS) propuesto en la Sección 6.1.4 con el Algoritmo Exterior (EXT) se obtiene un nuevo algoritmo que denominamos **Búsqueda en Entorno Variable Híbrida Exterior** (EHVNS, *Exterior Hybridized Variable Neighborhood Search*) y que reúne la eficacia de la Búsqueda en Entorno Variable y la flexibilidad que introduce el uso de soluciones α-potenciales en el Algoritmo Exterior. La combinación de los dos algoritmos propuestos se realiza intercalando la ejecución de ambos, utilizando el algoritmo HVNS como estrategia de intensificación para la obtención de buenos óptimos locales (con respecto a todas las estructuras de entornos) y el algoritmo EXT como estrategia de diversificación para la obtención de nuevas soluciones notablemente diferentes, aunque también de buena calidad, con las que continuar el proceso de búsqueda (lo cual se consigue con la búsqueda exterior, permitiendo la introducción paulatina de infactibilidades en la solución dada por la HVNS). En la Sección 7, dedicada a presentar la experiencia computacional realizada, se muestra cómo este algoritmo

combinado mejora notablemente los resultados previos publicados por otros autores en la literatura.

6.4. Algoritmo Genérico de Multiarranque con Intensificación

Los principales algoritmos heurísticos para el DTSPMS presentados en las secciones anteriores (HVNS, SAR, EXT, EHVNS) siempre toman como parámetros de entrada el *máximo tiempo de cómputo* disponible (necesario para el criterio de parada) y una *solución inicial* (necesaria para comenzar el proceso de búsqueda), además de los parámetros que sean necesarios para determinar el funcionamiento del algoritmo correspondiente (estructuras de entornos a utilizar, número de movimientos aleatorios que se realizan en una perturbación, temperaturas inicial y final, etc.). Al estar estructurados de esta manera, todos estos algoritmos se pueden utilizar fácilmente con varias soluciones iniciales y con distintos tiempos de ejecución para cada una de ellas, permitiendo incluso aplicar secuencialmente varios algoritmos o utilizar cierto tiempo de cómputo para realizar un proceso de intensificación sobre las mejores soluciones obtenidas a partir del conjunto de soluciones iniciales.

A continuación se presenta el pseudocódigo del denominado Algoritmo Genérico de Multiarranque con Intensificación (MSI, *Generic MultiStart and Intensification algorithm*), que permite aplicar cualquiera de los algoritmos heurísticos de mejora presentados anteriormente (HVNS, SAR, EXT, EHVNS) a un conjunto de varias soluciones iniciales. Dicho conjunto se va sucesivamente mejorando y actualizando, seleccionando en cada etapa las mejores soluciones obtenidas para tomarlas como soluciones iniciales en la etapa siguiente.

Algoritmo 27. Algoritmo Genérico de Multiarranque con Intensificación (MSI)

Parámetros de entrada

- σ: Número de etapas.
- $n_1, \cdots, n_\sigma$: Número de sol. iniciales consideradas en cada etapa ($n_1 > \cdots > n_\sigma$).
- $t_1, \cdots, t_\sigma$: Tiempo máximo de computación disponible para cada etapa.
- t^*: Tiempo máximo de computación disponible para la fase de Intensificación.
- MEJ: Algoritmo utilizado para mejorar cada solución inicial.
- *param*: Lista de parámetros necesarios para el algoritmo MEJ.

Parámetros de salida

- S^*: Mejor solución encontrada.

Pseudocódigo

1. *Generación de soluciones iniciales:* Construir un conjunto $\Gamma = \{S_1, \cdots, S_{n_1}\}$ formado por n_1 soluciones iniciales para el DTSPMS. Poner $j = 1$.
2. *Mejora inicial:* Hacer S=MEJ$(S, \frac{t_j}{n_j}, param)$ para cada $S \in \Gamma$.
3. *Selección*:
   - Si $j < \sigma$: poner $j = j+1$, particionar Γ en dos subconjuntos disjuntos $\Gamma = \Gamma_1 \cup \Gamma_2$ de manera que $|\Gamma_1| = n_j$ y $z(S) \leq z(\hat{S})\ \forall\ S \in \Gamma_1,\ \forall \hat{S} \in \Gamma_2$. Poner $\Gamma = \Gamma_1$ y volver al paso 2.
   - En otro caso: elegir $S^* \in \Gamma$ tal que $z(S^*) \leq z(S)\ \forall S \in \Gamma$.
4. *Intensificación:* Hacer S^*=MEJ(S^*,t^*,$param$).
5. *Final:* La mejor solución encontrada es S^*.

El Algoritmo Genérico Conjunto de Multiarranque con Intensificación (JMSI, *Joint Generic MultiStart and Intensification algorithm*) cuyo pseudocódigo se presenta a continuación permite aplicar secuencialmente varios algoritmos heurísticos de mejora, combinando sus efectos para la obtención de soluciones finales de mayor calidad.

Algoritmo 28. Algoritmo Genérico Conjunto de Multiarranque con Intensificación (JMSI)

Parámetros de entrada

- σ: Número de etapas.
- $n_1, \cdots, n_\sigma$: Número de soluciones iniciales consideradas en cada etapa.
- $t_1, \cdots, t_\sigma$: Tiempo máximo de computación disponible para cada etapa.
- t^*: Tiempo máximo de computación disponible para la fase de Intensificación.
- MEJ: Algoritmo utilizado para mejorar cada solución inicial en el MSI.
- *param*: Lista de parámetros necesarios para el algoritmo MEJ.
- ϱ: Número de mejoras sobre la solución dada por el algoritmo MSI.
- $\hat{t}_1, \cdots, \hat{t}_\varrho$: Tiempo máximo de computación disponible para cada mejora sobre la solución dada por el algoritmo MSI.

- $\text{MEJ}_1, \cdots, \text{MEJ}_\varrho$: Algoritmos utilizados para las mejoras sobre la solución dada por el algoritmo MSI.
- $param_1, \cdots, param_\varrho$: Listas de parámetros necesarios para los algoritmos de mejora.

Parámetros de salida

- S^*: Mejor solución encontrada.

Pseudocódigo

1. *Solución inicial mejorada:* Hacer S^*=MSI(σ,$n_1, \cdots, n_\sigma$,$t_1, \cdots, t_\sigma$,t^*,MEJ,$param$).
2. *Inicio*: Poner $i = 1$.
3. *Mejora:* Hacer $S^* = \text{MEJ}_i(S^*, \hat{t}_i, param_i)$.
4. *Parada:* Si $i < \varrho$, poner $i = i + 1$ y volver al paso 3. En otro caso, FIN. La mejor solución encontrada es S^*.

Los algoritmos heurísticos presentados en este capítulo se utilizarán para la obtención de los resultados computacionales presentados y comentados en el capítulo siguiente.

Capítulo 7

Resultados Computacionales

En este capítulo se comentan y relacionan los resultados computacionales obtenidos al aplicar los algoritmos propuestos en los capítulos anteriores a la resolución del DTSPMS. En la Sección 7.1 se introducen los conjuntos de instancias que se van a utilizar para realizar las pruebas computacionales, en la Sección 7.2 se resumen los mejores resultados publicados por otros autores en la literatura sobre el DTSPMS y en la Sección 7.3 se presentan algunos resultados sobre cotas inferiores obtenidos con el algoritmo CCA para instancias de pequeño tamaño. En la Sección 7.4 se presenta un estudio de calibración de los principales parámetros del algoritmo de Búsqueda en Entorno Variable Híbrida que sirve como modelo para los demás algoritmos heurísticos considerados y en la Sección 7.5 se recopilan y comentan los resultados obtenidos con los algoritmos de Búsqueda en Entorno Variable propuestos en el Capítulo 6. Los resultados del algoritmo Exterior introducido en la Sección 6.3 se muestran en la Sección 7.6, seguidos de los obtenidos con el algoritmo de Búsqueda en Entorno Variable Híbrida Exterior, presentados en la Sección 7.7. Por último, en la Sección 7.8 se comparan los mejores resultados obtenidos con las heurísticas propuestas en este trabajo con los mejores resultados publicados en la literatura por otros autores y en la Sección 7.9 se presenta un breve estudio de comparación entre dos implementaciones distintas de un algoritmo de Temple Simulado aplicado al DTSPMS.

El algoritmo CCA ha sido implementado en AMPL y ejecutado en un equipo con un procesador de 2000 Mhz, utilizando CPLEX 11 para la resolución de las Relajaciones por Precedencias, mientras que los algoritmos heurísticos han sido implementados en Fortran 95 y ejecutados en un equipo con un procesador de 1600 Mhz. Con el término *calidad* de una solución nos referiremos en lo que sigue a la lejanía de dicha solución con respecto a la mejor solución conocida, que se calculará dividiendo el coste de la solución dada entre el coste de la mejor solución conocida. A la desviación en porcentaje nos referiremos con el término *desviación* (o *gap*), que se calculará como $100(calidad - 1)$.

En todos los casos en que una sola solución inicial factible sea requerida ésta se calculará resolviendo el DTSPSS asociado (Sección 4.2.1.1). Si se requiere la generación de más de una solución inicial factible, la primera se calculará mediante dicho procedimiento y las demás mediante Generación Aleatoria Dirigida (Sección 4.2.1.2).

7.1. Conjuntos de instancias

Para la evaluación y comparación de los algoritmos utilizados y para realizar la calibración de sus parámetros se han considerado diversos conjuntos de instancias. Para evaluar los algoritmos propuestos se han utilizado los 3 conjuntos de 20 instancias con 12, 33 y 66 encargos utilizados en Petersen y Madsen (2009), que denominaremos *P12*, *P33* y *P66*, respectivamente. También se ha utilizado un conjunto de 20 instancias de tamaño 18, denominado *P18*, cuyas instancias fueron obtenidas eliminando los últimos 15 encargos de las instancias del conjunto *P33*. Los archivos de datos de estas instancias se pueden encontrar en la página web de Hanne L. Petersen, http://www.imm.dtu.dk/∼*hlp*.

Además de las instancias tomadas de Petersen y Madsen (2009), para evaluar el comportamiento de los algoritmos propuestos al ser aplicados a instancias de mayor tamaño se ha generado aleatoriamente otro conjunto de 20 instancias con 132 encargos, que se ha denominado *T132*. Los archivos de datos de estas instancias se pueden encontrar en la página web de Gregorio Tirado Domínguez, http://www.espaciotd.jazztel.es/dtspmsEn.htm.

Para realizar la calibración de parámetros de algunos de los algoritmos propuestos se han generado aleatoriamente 3 conjuntos de 10 instancias con 33, 66 y 132 encargos, denominados *C33*, *C66* y *C132*, respectivamente. Los archivos de datos de estas instancias también pueden encontrarse en http://www.espaciotd.jazztel.es/dtspmsEn.htm.

Todas las instancias utilizadas para la obtención de los resultados presentados en esta monografía, tanto las tomadas del trabajo de Petersen y Madsen (2009) como las introducidas aquí, fueron generadas de la misma forma: las localizaciones de recogida y entrega fueron seleccionadas aleatoriamente dentro del cuadrado 100×100, mientras que el depósito fue fijado en el centro del cuadrado (50,50) en todos los casos; el coste de viaje de una localización a otra se calcula, a partir de sus coordenadas, como la distancia Euclídea entre ambas redondeada al entero más próximo, siguiendo la convención de la TSPLIB, y a no ser que se indique lo contrario siempre se considerará que se dispone de 3 pilas con una capacidad máxima igual a la tercera parte del número de encargos.

Las dimensiones de las instancias utilizadas (33, 66, 132 encargos y 3 pilas independientes) fueron elegidas para representar lo más fielmente posible la situación real que originó el planteamiento del problema, en la cual los vehículos disponibles cargaban contenedores en los que se almacenaban 33 euro-palés colocados en 3 pilas de capacidad 11. Aunque

todos los resultados presentados en este capítulo, salvo en el caso de las instancias de 12 encargos y 2 pilas de capacidad 6 utilizadas en la Sección 7.3, son para instancias con 3 pilas y capacidad máxima lo más ajustada posible (el número de encargos dividido por el número de pilas), también se realizaron pruebas con instancias con 2 ó 4 pilas, obteniendo resultados similares. La utilización de instancias de tamaño mayor (doble o cuádruple) es necesaria para evaluar el comportamiento de los algoritmos cuando el número de encargos crece, y se justifica de forma sencilla considerando la utilización de contenedores con mayor altura que permiten el almacenaje de varios palés unos encima de otros en la misma pila. Por último, también se consideran instancias con 12 encargos porque son las instancias de mayor tamaño que pudieron resolverse de forma exacta.

7.2. Mejores resultados previos

En esta sección se presentan brevemente los mejores resultados publicados en la literatura por otros autores para las instancias con 12, 33 y 66 encargos consideradas en la sección anterior (conjuntos *P12*, *P33* y *P66*), los cuales han sido obtenidos por Petersen y Madsen (2009) utilizando un algoritmo de Búsqueda en Entornos Grandes (LNS). No se presentan resultados para las instancias con 132 encargos porque no existen resultados previos para instancias con más de 66 encargos.

El algoritmo LNS obtuvo las soluciones óptimas de todas las instancias del conjunto *P12*, con tan sólo 12 encargos, con un tiempo máximo de cómputo de 10 segundos por instancia. Los resultados obtenidos por dicho algoritmo para las instancias de los conjuntos *P33* y *P66* se presentan, respectivamente, en las Tablas 7.1 y 7.2, que tienen el mismo formato: la primera columna indica el nombre de cada instancia, la segunda una cota inferior obtenida resolviendo los TSPs de recogida y entrega de forma independiente, en la tercera se presentan los costes de las mejores soluciones encontradas por el algoritmo LNS y por último las columnas cuarta y quinta contienen la calidad de las soluciones obtenidas por el algoritmo LNS utilizando, respectivamente, 10 segundos y 3 minutos de tiempo de cómputo.

7.3. Cotas inferiores para problemas de pequeño tamaño

En esta sección se presentan los resultados obtenidos al aplicar al DTSPMS el Algoritmo de Cortes con Conflictos (CCA) propuesto en la Sección 3.2.3. En Petersen et al. (2008) se propone un algoritmo exacto para el DTSPMS con el que se consiguen resolver instancias de hasta 15 encargos con 3 pilas y algunas de 20 encargos con 4 pilas, siendo éstas las instancias del DTSPMS de mayor tamaño que han podido ser resueltas de forma exacta. Sin embargo, no se consiguen resolver de forma exacta las instancias con 12 encargos y 2

Instancia	LB	Best	10 seg	3 min
R00	911	1063	1.04	1.01
R01	875	1032	1.04	1.01
R02	935	1065	1.04	1.01
R03	961	1100	1.06	1.01
R04	937	1052	1.05	1.02
R05	900	1008	1.03	1.01
R06	998	1110	1.06	1.02
R07	963	1105	1.05	1.01
R08	978	1109	1.04	1.01
R09	976	1091	1.04	1.01
R10	901	1016	1.05	1.00
R11	892	1001	1.06	1.01
R12	984	1109	1.04	1.01
R13	956	1084	1.04	1.01
R14	879	1034	1.03	1.00
R15	985	1142	1.04	1.01
R16	967	1093	1.02	1.00
R17	946	1073	1.04	1.00
R18	1008	1118	1.05	1.01
R19	938	1089	1.03	1.01
Media			1.04	1.01

Tabla 7.1: LNS con 33 encargos

Instancia	LB	Best	10 seg	3 min
R00-66	1237	1594	1.19	1.07
R01-66	1257	1600	1.20	1.08
R02-66	1295	1576	1.20	1.12
R03-66	1290	1631	1.14	1.06
R04-66	1295	1611	1.18	1.09
R05-66	1204	1528	1.18	1.07
R06-66	1294	1651	1.17	1.07
R07-66	1307	1653	1.17	1.08
R08-66	1297	1607	1.18	1.07
R09-66	1276	1598	1.18	1.08
R10-66	1339	1702	1.17	1.09
R11-66	1268	1575	1.19	1.08
R12-66	1295	1652	1.19	1.10
R13-66	1275	1617	1.19	1.10
R14-66	1245	1611	1.21	1.09
R15-66	1228	1608	1.19	1.10
R16-66	1356	1725	1.16	1.07
R17-66	1274	1627	1.21	1.10
R18-66	1328	1671	1.18	1.08
R19-66	1256	1635	1.17	1.09
Media			1.18	1.08

Tabla 7.2: LNS con 66 encargos

pilas con capacidad 6 ni las instancias con 18 encargos y 3 pilas de capacidad 6, ya que al tener menor número de pilas el problema está más restringido y su resolución exacta es más complicada. Para evaluar el comportamiento del algoritmo CCA en estas instancias no resueltas de forma exacta se ha aplicado dicho algoritmo a las instancias del conjunto *P12*, permitiendo el uso de sólo 2 pilas de capacidad 6 (denominadas R00-12 - R19-12), y a las instancias del conjunto *P18*, permitiendo el uso de 3 pilas de capacidad 6 (denominadas R00-18 - R19-18). Los resultados sobre las instancias de *P12* se presentan en la Tabla 7.3 y los de las instancias de *P18* en la Tabla 7.4. Algunos de estos resultados previos ya han sido presentados en una conferencia internacional (Ceselli et al., 2009).

La primera columna de las Tablas 7.3 y 7.4 contiene los nombres de las instancias consideradas y la segunda la cota inferior (LB) obtenida con el algoritmo CCA con un tiempo de cómputo máximo de una hora, que es mismo límite de tiempo utilizado en Petersen et al. (2008) para la resolución de las mismas instancias. En la tercera columna se muestran las mejores cotas superiores (UB) obtenidas con las heurísticas propuestas en este trabajo para estas instancias, que debido a su pequeño tamaño son las mismas para todas las heurísticas y probablemente coincidan en su mayoría con los valores óptimos. En

la cuarta columna se presenta la diferencia en porcentaje (desviación o gap) entre la cota superior y la inferior, calculada como $100(\frac{UB-LB}{LB})$, y en la quinta el tiempo de cómputo (en segundos) requerido por el algoritmo CCA para cada instancia, indicando con - que se ha alcanzado el tiempo máximo de una hora.

Instancia	LB	UB	GAP	T(s)
R00-12	716	726	1.40	440
R01-12	741	741	**0.00**	869
R02-12	651	660	1.38	-
R03-12	690	690	**0.00**	120
R04-12	659	659	**0.00**	-
R05-12	620	631	1.77	-
R06-12	789	793	0.51	992
R07-12	589	593	0.68	1696
R08-12	749	749	**0.00**	-
R09-12	686	692	0.87	133
R10-12	662	663	0.15	-
R11-12	622	625	0.48	911
R12-12	741	741	**0.00**	183
R13-12	683	694	1.61	104
R14-12	680	680	**0.00**	2148
R15-12	624	628	0.64	-
R16-12	610	610	**0.00**	1505
R17-12	761	780	2.50	-
R18-12	735	735	**0.00**	185
R19-12	782	789	0.90	1785
Media			**0.64**	1813

Tabla 7.3: CCA con 12 encargos y 2 pilas

Instancia	LB	UB	GAP	T(s)
R00-18	778	803	3.21	-
R01-18	763	791	3.67	-
R02-18	691	706	2.17	-
R03-18	793	828	4.41	-
R04-18	735	759	3.27	-
R05-18	791	804	1.64	-
R06-18	839	866	3.22	-
R07-18	727	752	3.44	-
R08-18	841	864	2.73	-
R09-18	764	774	1.31	266
R10-18	760	787	3.55	-
R11-18	732	743	1.50	-
R12-18	780	803	2.95	-
R13-18	789	813	3.04	-
R14-18	763	792	3.80	-
R15-18	774	801	3.49	-
R16-18	731	747	2.19	-
R17-18	799	815	2.00	-
R18-18	804	828	2.99	-
R19-18	822	833	1.34	-
Media			**2.80**	3431

Tabla 7.4: CCA con 18 encargos y 3 pilas

En la Tabla 7.3 se observa que para 8 de las 20 instancias de tamaño 12 consideradas se obtiene la solución óptima (GAP=0 %), mientras que el GAP medio entre todas ellas es inferior al 1 %. A pesar de tener las mismas dimensiones nótese la gran diferencia existente entre el tiempo de cómputo utilizado por las distintas instancias, que varía desde dos minutos hasta el límite máximo de una hora.

Para las instancias de tamaño 18 no se consigue ninguna solución óptima, como puede observarse en la Tabla 7.4, pero se obtiene un GAP medio menor del 3 %. En todas las instancias menos una se alcanza el tiempo máximo de cómputo, lo que indica que estas instancias son sensiblemente más difíciles de resolver que las de tamaño 12.

7.4. Calibración de parámetros en las heurísticas

En esta sección se presenta un estudio de calibración de los principales parámetros del algoritmo de Búsqueda en Entorno Variable Híbrida para ser aplicado a instancias del DTSPMS con distinto número de encargos en combinación con el algoritmo Genérico de Multiarranque con Intensificación. Se ha elegido el HVNS porque es el algoritmo más complejo, en cuanto a su funcionamiento y a la cantidad de parámetros a calibrar, tanto dentro del grupo de algoritmos de Búsqueda en Entorno Variable como en comparación con los demás algoritmos considerados; la calibración de parámetros del resto de algoritmos es mucho más sencilla y se ha realizado de forma análoga.

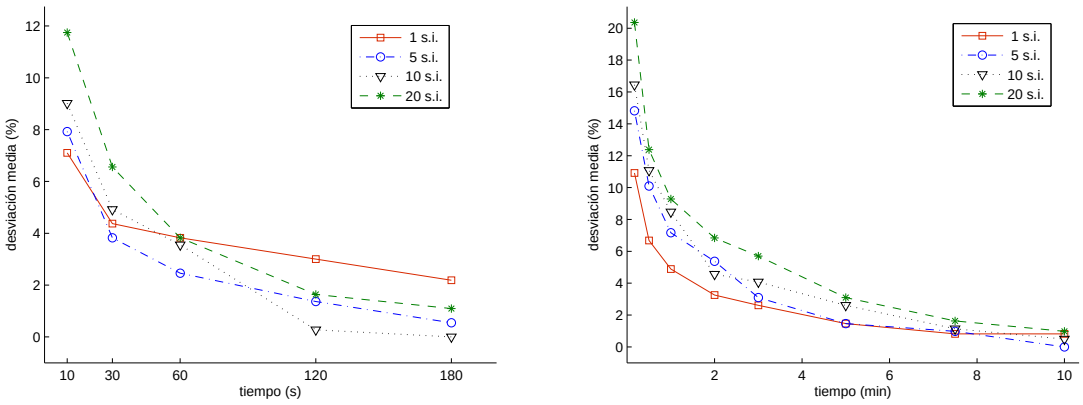
Para aplicar todas las heurísticas propuestas al DTSPMS utilizaremos como base la estructura dada por el algoritmo Genérico de Multiarranque con Intensificación (MSI) con una sola etapa $\sigma = 1$, de manera que lo que se hace es dedicar cierta cantidad del tiempo de cómputo disponible para aplicar el algoritmo elegido (HVNS, SA, EXT, etc., según el caso) a las soluciones iniciales generadas (multiarranque) y posteriormente dedicar el resto del tiempo de cómputo a aplicar el mismo algoritmo sobre la mejor solución obtenida a partir de las distintas soluciones iniciales (intensificación). Así, además de los parámetros del algoritmo de mejora que se considere (HVNS, SA, EXT, etc.) habrá que determinar el valor de los siguientes parámetros relacionados con la estructura MSI:

- m_I: Número de soluciones iniciales consideradas.
- $pInt$: Porcentaje del tiempo de cómputo disponible que se dedica al proceso de Intensificación (mejora de la mejor solución obtenida tras el tratamiento de las soluciones iniciales).

A continuación se presentan algunas gráficas que muestran los resultados obtenidos al aplicar el algoritmo HVNS con distintos números de soluciones iniciales y distinto porcentaje de tiempo dedicado a la intensificación a la resolución de los conjuntos de instancias *C33*, *C66* y *C132*, generadas para ser utilizadas en la calibración de los parámetros más importantes de los algoritmos considerados. Estos resultados nos permiten calibrar los parámetros relativos a la estructura MSI del algoritmo HVNS, pero también sirven de guía para calibrar dichos parámetros para los casos de otras heurísticas, ya que su comportamiento es similar.

En las gráficas de las Figuras 7.1 y 7.2 el tiempo de cómputo en segundos se representa en el eje horizontal y el desvío en porcentaje de la mejor solución conocida se representa en el eje vertical. Además, nótese que para representar adecuadamente la información contenida en dichas gráficas la escala utilizada en el eje vertical es diferente en cada caso.

En la Figura 7.1 se presentan los resultados obtenidos para el conjunto *C66* de instancias con 66 encargos (7.1a) y el conjunto *C132* de instancias con 132 encargos (7.1b) utilizando 1, 5, 10 ó 20 soluciones iniciales. Para el primero de estos conjuntos de instancias se observa que con 10 segundos de tiempo de cómputo el uso de una sola solución inicial es la mejor opción, pues la calidad de las soluciones obtenidas disminuye cada vez que se aumenta el número de soluciones iniciales; sin embargo, si el tiempo de cómputo disponible aumenta (30, 60 segundos), se observa que utilizando 5 soluciones iniciales se obtienen mejores resultados que utilizando sólo una, mientras que si se dispone de 2-3 minutos de tiempo de cómputo la mejor elección es de 10 soluciones iniciales, y después 5, 20 y por último 1, que da lugar a los peores resultados. A la vista de esto concluimos que, como era de esperar, el número de soluciones iniciales debe aumentarse proporcionalmente al tiempo de cómputo disponible.



(a) *C66* (b) *C132*

Figura 7.1: Resultados de calibración: distinto número de soluciones iniciales

En las instancias del conjunto *C132*, con 132 encargos, se observa un comportamiento similar, aunque ahora el uso de varias soluciones iniciales no resulta útil si no se consideran tiempos de cómputo mayores, de al menos 5 minutos. Esto indica que el número de soluciones iniciales puede incrementarse rápidamente para instancias de tamaño medio (33-66 encargos) pero no para instancias sensiblemente mayores (132 encargos), para las cuales es necesario dedicar un mayor tiempo de cómputo a cada solución inicial para la obtención de buenos óptimos locales. Esto nos permite concluir que el número de soluciones iniciales debe ser determinado en cada caso, aumentando con el tiempo de cómputo total disponible y disminuyendo con el tamaño de las instancias a resolver.

La Figura 7.2 muestra los resultados obtenidos para el conjunto *C66* de instancias con 66 encargos utilizando el 0 %, 10 %, 20 % ó 50 % del tiempo de cómputo disponible para realizar la intensificación sobre la mejor solución encontrada en la fase de multiarranque.

Se observa claramente que los peores resultados se obtienen no realizando ningún tipo de intensificación, mientras que no existen diferencias significativas entre las otras 3 elecciones. Para instancias de tamaños diferentes (33 ó 132 encargos) se obtienen resultados similares, lo que nos permite concluir que la intensificación es útil y por tanto $pInt$ debe ser estrictamente mayor que cero. Como no se observan grandes diferencias entre los porcentajes distintos de 0 considerados optamos por tomar un valor intermedio y dedicar el 20 % del tiempo de cómputo disponible a la intensificación.

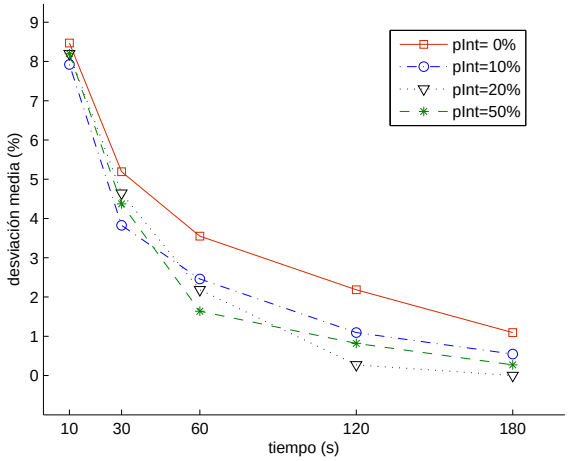


Figura 7.2: Resultados de calibración: Porcentaje de tiempo dedicado a la intensificación

Además de los parámetros m_I y $pInt$, que determinan la estructura de multiarranque e intensificación del algoritmo, habrá que determinar en cada caso los parámetros que necesite el algoritmo de mejora elegido (HVNS, SA, EXT, etc.). En el caso del HVNS, elegido para nuestro estudio, los principales parámetros a determinar son los siguientes:

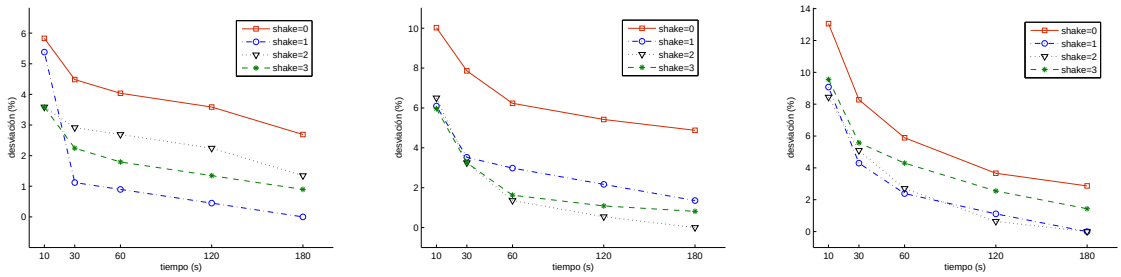
- α: Tamaño adicional permitido en las pilas.
- *reinicio*: Constante booleana que indica si se reinicia el recorrido de los entornos al conseguir mejorar la solución (T) o no (F).
- $\Delta = \{\Delta_k,\ k = 1, \cdots, e_1\}$: Conjunto finito de estructuras de entornos sobre X para perturbar la solución.
- $\Omega = \{\Omega_k,\ k = 1, \cdots, e_2\}$: Conjunto finito de estructuras de entornos sobre X para el descenso (VND).
- *shake*: Número de agitaciones que se realizan dentro del mismo entorno al realizar una perturbación de la solución.
- θ: Número de ordenaciones distintas de $\{\Omega_k\}$ que se consideran para la realización del descenso.

- $R = \{R_1, \cdots, R_\theta\}$: Conjunto de θ permutaciones de $\{1, \cdots, e_2\}$.
- $P = \{P_1, \cdots, P_\theta\}$: Probabilidades con las que se eligen los distintos órdenes $\{R_1, \cdots, R_\theta\}$.
- $itmax$: Número máximo de iteraciones permitidas sin mejorar la solución actual.
- *pcost*: Porcentaje máximo de desvío en coste con respecto a la mejor solución conocida.
- l_{tabu}: Longitudes de las listas tabú.

La influencia del tamaño adicional permitido en las pilas de las soluciones y la consideración de soluciones α-factibles se estudiará con más detalle en las secciones siguientes, por lo que dejaremos para más adelante la discusión sobre el parámetro α. El parámetro *reinicio* se fijará siempre a T, tanto en el algoritmo HVNS con en todos los demás algoritmos de Búsqueda en Entorno Variable donde aparezca, pues se ha observado que es conveniente reiniciar la búsqueda desde el primer entorno considerado cada vez que se mejora la solución actual. Los conjuntos de estructuras de entornos a utilizar por todos los algoritmos de Búsqueda en Entorno Variable (VND, GVNS, HVNS) serán Ω = {RS, CS, ISS, R, 3-RP, 4-CSP}, para la búsqueda local, y Δ = {RS, CS, ISS, R} para la perturbación, en caso de utilizarse. Para la búsqueda local se utilizarán todas las estructuras de entornos propuestas en 4.3.1-4.3.7 salvo r-SP, ya que en su lugar se utiliza r-CSP, y se toman valores intermedios $r = 3$ para el entorno RP y $r = 4$ para el entorno CSP para generar movimientos efectivos pero no demasiado complejos. Para la perturbación se excluyen los entornos 3-RP y 4-CSP por simplicidad, ya que se trata de realizar movimientos aleatorios sencillos en los que la consideración de permutaciones no es necesaria.

En la Figura 7.3, donde al igual que en las Figuras 7.1 y 7.2 el tiempo de cómputo en segundos se representa en el eje horizontal y el desvío en porcentaje de la mejor solución conocida se representa en el eje vertical, se muestran los resultados obtenidos para distintos valores del parámetro $shake = 0, 1, 2, 3$ para los 3 conjuntos de 10 instancias considerados para la calibración de parámetros (*C33*, *C66* and *C132*). El parámetro *shake* determina cuánto se perturba un óptimo local antes de proseguir a partir de él el proceso de búsqueda, de manera que cuanto mayor sea el valor del parámetro más operaciones de perturbación se realizarán y más se alejará la siguiente solución del óptimo local anterior. Se observa claramente que no realizar perturbación alguna ($shake = 0$) produce los peores resultados en todos los conjuntos de instancias, mientras que no existen diferencias significativas entre realizar 1, 2 ó 3 perturbaciones en cada iteración. Por tanto llegamos a la conclusión de que *shake* debe ser con certeza mayor que cero y decidimos tomar un valor intermedio y fijar $shake = 2$.

En las pruebas computacionales realizadas sobre las instancias de calibración utilizando distintos órdenes en el recorrido de las estructuras de entornos y asociándolos a distintas



(a) *C33* (b) *C66* (c) *C132*

Figura 7.3: Resultados de calibración: parámetro *shake*=0,1,2,3

probabilidades no se han observado diferencias significativas, por lo que se ha decidido fijar el orden usual de menor a mayor tamaño en las estructuras utilizadas: 3-RP, RS, ISS, CS, R, 4-CSP. El máximo número de iteraciones permitido sin mejorar la solución actual resulta efectivo para las instancias consideradas si está entre 5 y 30, por lo que ha tomado una elección intermedia $itmax = 20$, y también se ha fijado la máxima desviación de coste permitida a $pcost = 30\,\%$ tras observar comportamientos similares del algoritmo para distintos valores próximos él. Por último, tras no observar cambios significativos al incluir listas tabú de distintas longitudes y con la intención de simplificar al máximo el algoritmo utilizado, se ha decidido prescindir de ellas en este estudio y fijar $l_{tabu} = 0$.

7.5. Búsqueda en Entorno Variable

En esta sección se presentan los resultados computacionales obtenidos aplicando los algoritmos VND, GVNS y HVNS introducidos en la Sección 6.1 a la resolución de las instancias de los conjuntos *P12*, *P33*, *P66* y *T132*. Todos ellos se ejecutan siguiendo la estructura de multiarranque e intensificación del algoritmo MSI y utilizando los parámetros elegidos según el análisis presentado en la Sección 7.4.

El algoritmo VND para cuando la solución actual es un óptimo local con respecto a todas las estructuras de entornos consideradas, siendo despreciable el tiempo de cómputo necesario para su ejecución al ser aplicado a todas las instancias consideradas. Además, dicho algoritmo es determinístico, de manera que fijados los parámetros siempre se obtienen las mismas soluciones en todas las ejecuciones del mismo. Por el contrario, el criterio de parada para los algoritmos GVNS y HVNS es el tiempo de cómputo disponible, y la aleatoriedad que se introduce en ellos para diversificar el proceso de búsqueda hace que sean no determinísticos y por tanto se puedan obtener resultados diferentes en ejecuciones distintas. Para que los resultados presentados sean más representativos, todos los resultados obtenidos con los algoritmos GVNS y HVNS incluidos en esta sección son resultados medios

de tres ejecuciones realizadas con distintas semillas iniciales.

En la Sección 7.5.1 se presentan los resultados obtenidos utilizando exclusivamente soluciones 0-factibles en el proceso de búsqueda, mientras que en la Sección 7.5.2 se presentan los resultados obtenidos permitiendo la utilización de soluciones intermedias α-factibles con $\alpha > 0$. En la Sección 7.5.3 se recopilan los mejores resultados obtenidos con los algoritmos de Búsqueda en Entorno Variable para cada conjunto de instancias considerado y, para finalizar, en la Sección 7.5.4 se presenta un estudio del impacto de las estructuras de entornos utilizadas en el comportamiento del algoritmo HVNS.

7.5.1. Búsqueda con soluciones 0-factibles

En las Secciones 7.5.1.1-7.5.1.4 se presentan los resultados obtenidos aplicando los algoritmos VND, GVNS y HVNS a las instancias de los conjuntos *P12*, *P33*, *P66* y *T132* *utilizando exclusivamente soluciones* 0-*factibles*, es decir, con $\alpha = 0$.

Las Tablas 7.5-7.7 presentadas en las Secciones 7.5.1.2-7.5.1.4 siguen la misma estructura, que se detalla a continuación: la primera columna contiene los nombres de las instancias consideradas, la segunda el valor de una cota inferior obtenida resolviendo de forma independiente los dos TSPs inducidos en las redes de recogida y entrega y la tercera el coste de las mejores soluciones conocidas para cada instancia; en la cuarta columna se muestra la calidad de las soluciones obtenidas por el algoritmo VND, mientras que en las cuatro siguientes se presenta la calidad media de las soluciones obtenidas por los algoritmos GVNS y HVNS, respectivamente, con 10 segundos de tiempo de cómputo en las dos primeras y con 3 minutos en las dos últimas. Finalmente, en la última fila se indican los resultados medios para cada algoritmo.

7.5.1.1. Instancias con 12 encargos

Las instancias más pequeñas consideradas en este trabajo (conjunto *P12*) tienen 12 encargos porque éstas son las instancias más grandes que han podido ser resueltas de forma exacta utilizando software comercial, y por tanto es el mayor tamaño para el cual los resultados obtenidos por las heurísticas se pueden comparar directamente con las soluciones óptimas. El conjunto *P12*, formado por 20 instancias de tamaño 12, fue generado a partir del conjunto *P33* de 20 instancias de tamaño 33 eliminando los últimos 21 encargos de cada instancia (Petersen y Madsen, 2009).

Las soluciones obtenidas por el algoritmo VND son, en media, un 12-14 % peores que las óptimas, no obteniéndose la solución óptima en ninguna de las instancias consideradas. Sin embargo nótese que se trata de un algoritmo muy sencillo para cuya ejecución se emplea un tiempo de cómputo despreciable.

El algoritmo HVNS ofrece resultados notablemente mejores y la desviación media baja al 0.2 % con un tiempo de cómputo máximo de sólo 1 segundo. Para 10 de las 20 instancias consideradas se obtiene la solución óptima en las 3 ejecuciones realizadas, mientras que en 18 de las 20 se obtiene la solución óptima en al menos 2 de las 3 ejecuciones realizadas. Sólo hay 2 instancias para las cuales no se encuentra la solución óptima en ninguna de las ejecuciones del algoritmo, pero en esos casos la desviación con respecto a la solución óptima es siempre menor del 1 %. Con un tiempo de cómputo de 5 segundos ya se obtienen las soluciones óptimas de todas las instancias en todas las ejecuciones del algoritmo.

En este caso de instancias de pequeño tamaño el algoritmo GVNS tiene un comportamiento muy parecido al HVNS, obteniéndose con ambos algoritmos resultados similares a los mejores resultados previos para este tipo de instancias, presentados en la Sección 7.2, según los cuales se obtenían las soluciones óptimas con 10 segundos de tiempo de cómputo por instancia.

7.5.1.2. Instancias con 33 encargos

En la Tabla 7.5 se presentan los resultados obtenidos al aplicar los algoritmos VND, GVNS y HVNS a las instancias del conjunto *P33*, formado por 20 instancias con 33 encargos denominadas R00-R19.

Las soluciones proporcionadas por el algoritmo VND son considerablemente peores que las mejores conocidas (un 42 % de media), aunque hay que tener siempre en cuenta que el tiempo de cómputo requerido por este algoritmo es totalmente despreciable. El algoritmo GVNS proporciona una mejora notable (desviación media del 12 % en 10 segundos y del 8.3 % en 3 minutos), pero el algoritmo que proporciona los mejores resultados es el algoritmo híbrido HVNS, específicamente diseñado para el DTSPMS, con una desviación media del 6 % en 10 segundos y del 2.2 % en 3 minutos.

7.5.1.3. Instancias con 66 encargos

En la Tabla 7.6 se presentan los resultados obtenidos al aplicar los algoritmos VND, GVNS y HVNS a las instancias del conjunto *P66*, formado por 20 instancias con 66 encargos denominadas R00-66 - R19-66. Los resultados contenidos en esta tabla son similares a los de la Tabla 7.5: las soluciones proporcionadas por el algoritmo VND están lejos de las mejores conocidas (un 49 % de media) y el algoritmo GVNS mejora notablemente las soluciones dadas por el algoritmo VND disminuyendo la desviación media a menos de la mitad; sin embargo, de nuevo el algoritmo con mejores resultados es el diseñado específicamente para el DTSPMS, el HVNS, con una desviación media del 23.8 % en 10 segundos y del 12.6 % en 3 minutos.

				10 seg		3 min	
Problema	LB	Best	VND	GVNS	HVNS	GVNS	HVNS
R00	911	1063	1.451	1.121	1.101	1.099	1.032
R01	875	1032	1.422	1.125	1.059	1.085	1.010
R02	935	1065	1.415	1.094	1.072	1.084	1.030
R03	961	1100	1.415	1.156	1.035	1.085	1.033
R04	937	1052	1.434	1.138	1.077	1.097	1.055
R05	900	1008	1.377	1.123	1.073	1.088	1.026
R06	998	1110	1.423	1.094	1.037	1.083	1.000
R07	963	1105	1.405	1.114	1.061	1.094	1.024
R08	978	1109	1.399	1.114	1.080	1.069	1.046
R09	976	1091	1.414	1.094	1.049	1.062	1.013
R10	901	1016	1.461	1.156	1.029	1.094	1.003
R11	892	1001	1.327	1.128	1.056	1.078	1.056
R12	984	1109	1.434	1.118	1.078	1.083	1.002
R13	956	1084	1.428	1.116	1.051	1.067	1.014
R14	879	1034	1.442	1.130	1.049	1.082	1.015
R15	985	1142	1.410	1.102	1.044	1.076	1.000
R16	967	1093	1.397	1.122	1.045	1.049	1.012
R17	946	1073	1.466	1.108	1.071	1.091	1.023
R18	1008	1118	1.410	1.119	1.077	1.102	1.048
R19	938	1089	1.460	1.119	1.062	1.084	1.006
Media			1.419	1.120	**1.060**	1.083	**1.022**

Tabla 7.5: VNS con soluciones 0-factibles (instancias 33 encargos)

7.5.1.4. Instancias con 132 encargos

En la Tabla 7.7 se presentan los resultados obtenidos al aplicar los algoritmos VND, GVNS y HVNS a las instancias del conjunto *T132*, formado por 20 instancias con 132 encargos denominadas R00-132 - R19-132. De nuevo los resultados de esta tabla siguen la misma línea que los de las Tablas 7.5 y 7.6: el VND proporciona soluciones muy alejadas de las mejores conocidas y el HVNS es el algoritmo con mejor comportamiento, seguido de cerca por el algoritmo GVNS. En el caso de las instancias con 132 encargos la diferencia entre los algoritmos GVNS y HVNS no es tan grande como en los casos con 33 y 66 encargos (desviaciones medias de 32.3 % frente a 32.0 % y 26 % frente a 23.8 %).

7.5.2. Búsqueda con soluciones α-factibles

En las Secciones 7.5.1.1-7.5.1.4 se presentan los resultados obtenidos aplicando los algoritmos VND, GVNS y HVNS a las instancias de los conjuntos *P12*, *P33*, *P66* y *T132*

				10 seg		3 min	
Problema	LB	Best	VND	GVNS	HVNS	GVNS	HVNS
R00-66	1237	1594	1.516	1.218	1.237	1.215	1.118
R01-66	1257	1600	1.472	1.280	1.225	1.221	1.164
R02-66	1295	1576	1.517	1.296	1.273	1.235	1.173
R03-66	1290	1630	1.480	1.258	1.240	1.218	1.111
R04-66	1295	1611	1.528	1.263	1.233	1.227	1.129
R05-66	1204	1534	1.493	1.229	1.220	1.199	1.091
R06-66	1294	1651	1.504	1.268	1.226	1.160	1.139
R07-66	1307	1653	1.433	1.278	1.215	1.195	1.120
R08-66	1297	1607	1.482	1.212	1.238	1.212	1.145
R09-66	1276	1598	1.466	1.223	1.206	1.198	1.133
R10-66	1339	1702	1.464	1.247	1.194	1.198	1.103
R11-66	1268	1575	1.502	1.228	1.243	1.190	1.143
R12-66	1295	1646	1.495	1.207	1.238	1.207	1.100
R13-66	1275	1616	1.454	1.244	1.263	1.207	1.134
R14-66	1245	1610	1.504	1.250	1.260	1.213	1.129
R15-66	1228	1604	1.503	1.258	1.271	1.168	1.133
R16-66	1356	1720	1.463	1.251	1.201	1.213	1.095
R17-66	1274	1627	1.538	1.275	1.269	1.213	1.134
R18-66	1328	1671	1.479	1.251	1.239	1.196	1.107
R19-66	1256	1635	1.472	1.259	1.272	1.209	1.124
Media			1.488	1.250	**1.238**	1.205	**1.126**

Tabla 7.6: VNS con soluciones 0-factibles (instancias 66 encargos)

utilizando soluciones α-factibles con $\alpha > 0$, es decir, permitiendo temporalmente la utilización de cierta capacidad extra en el tamaño de las pilas introduciendo la infactibilidad IC en el proceso de búsqueda. Se espera que con esta estrategia se flexibilice el proceso de búsqueda y se puedan mejorar los resultados presentados en la Sección 7.5.1. Nótese que, a pesar de la introducción de soluciones α-factibles, las soluciones finales obtenidas a las cuales corresponden los resultados presentados son siempre soluciones 0-factibles.

Para cada conjunto de instancias (salvo *P12*) se presenta una gráfica en la que se comparan los resultados obtenidos con soluciones α-factibles para distintos valores de $\alpha = 0, 1, 2, 3$. Posteriormente, con el objetivo de observar con detalle el comportamiento de los algoritmos con el mismo tipo de soluciones en las distintas instancias consideradas, se presentan los resultados obtenidos para cada grupo de instancias utilizando soluciones 2-factibles. Se eligen este tipo de soluciones porque utilizan un tamaño extra intermedio entre los valores probados. Estos resultados se presentan en las Tablas 7.8-7.10, que siguen la misma estructura que las Tablas 7.5.1.2-7.5.1.4 de la sección anterior.

				10 seg		3 min	
Problema	LB	Best	VND	GVNS	HVNS	GVNS	HVNS
R00-132	1741	2591	1.525	1.344	1.298	1.281	1.225
R01-132	1789	2645	1.505	1.358	1.301	1.267	1.245
R02-132	1778	2639	1.535	1.302	1.299	1.262	1.204
R03-132	1807	2752	1.539	1.290	1.297	1.250	1.207
R04-132	1747	2603	1.469	1.312	1.315	1.246	1.237
R05-132	1753	2616	1.544	1.337	1.348	1.252	1.296
R06-132	1742	2576	1.496	1.333	1.339	1.280	1.228
R07-132	1712	2615	1.494	1.378	1.366	1.262	1.224
R08-132	1747	2638	1.506	1.323	1.335	1.254	1.241
R09-132	1708	2554	1.476	1.290	1.299	1.226	1.245
R10-132	1799	2646	1.481	1.336	1.324	1.253	1.239
R11-132	1779	2632	1.475	1.292	1.296	1.244	1.198
R12-132	1718	2555	1.530	1.332	1.326	1.280	1.295
R13-132	1765	2659	1.509	1.291	1.314	1.242	1.212
R14-132	1739	2605	1.516	1.296	1.316	1.257	1.251
R15-132	1729	2626	1.495	1.329	1.335	1.255	1.239
R16-132	1717	2534	1.513	1.340	1.330	1.289	1.277
R17-132	1694	2569	1.534	1.359	1.330	1.270	1.286
R18-132	1790	2652	1.455	1.302	1.291	1.252	1.219
R19-132	1764	2644	1.511	1.312	1.341	1.271	1.187
Media			1.505	1.323	**1.320**	1.260	**1.238**

Tabla 7.7: VNS con soluciones 0-factibles (instancias 132 encargos)

7.5.2.1. Instancias con 12 encargos

En las instancias del conjunto *P12*, con sólo 12 encargos, las diferencias son mínimas debido a su pequeño tamaño: la desviación media de las soluciones obtenidas con el algoritmo VND se mantiene alrededor del 10 % y con los algoritmos GVNS y HVNS se consiguen soluciones óptimas para todas las instancias en menos de 5 segundos de tiempo de cómputo.

7.5.2.2. Instancias con 33 encargos

En la gráfica de la Figura 7.4 se comparan los resultados obtenidos para las instancias del conjunto *P33* con los algoritmos GVNS y HVNS utilizando soluciones α-factibles para distintos valores de $\alpha = 0, 1, 2, 3$. En el eje horizontal se representa el número de unidades extra permitidas en el tamaño máximo de las pilas de las soluciones intermedias ($\alpha = 0, 1, 2, 3$) y en el eje vertical la desviación media de las soluciones obtenidas por cada algoritmo para cada valor del parámetro α. Los puntos marcados con $\square$ hacen referencia a

las soluciones obtenidas con el algoritmo GVNS en 10 segundos, los marcados con ∘ a las obtenidas con el algoritmo GVNS en 3 minutos, los marcados con ▽ a las obtenidas con el algoritmo HVNS en 10 segundos y los marcados con * a las obtenidas con el algoritmo HVNS en 3 minutos.

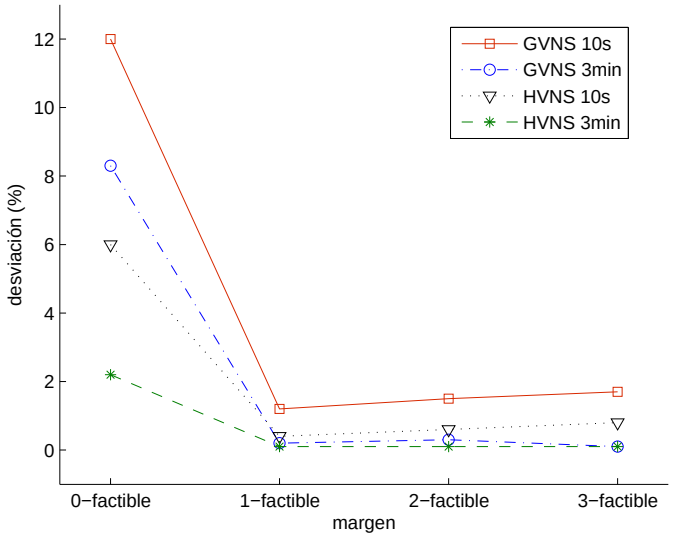


Figura 7.4: HVNS y GVNS con soluciones α-factibles, $\alpha = 0, 1, 2, 3$ (instancias 33 encargos)

Se observa que para ambos algoritmos y ambos tiempos de cómputo la mayor desviación media siempre se obtiene con $\alpha = 0$, es decir, no utilizando ninguna unidad extra en el tamaño de las pilas de las soluciones intermedias, mientras que no se observan diferencias significativas entre el uso de 1, 2 ó 3 unidades de tamaño extra. Esto muestra claramente la importancia de la introducción de las soluciones α-factibles, con las cuales se obtienen resultados notablemente mejores en todos los casos. En el caso del algoritmo GVNS, con 10 segundos de tiempo de cómputo la desviación media baja del 12 % a menos del 2 %, y con 3 minutos baja del 8 % a valores menores del 1 %. En el caso del algoritmo HVNS se observa un comportamiento similar, bajando la desviación media del 6 % a menos del 1 % utilizando 10 segundos de tiempo de cómputo y bajando de algo más del 2 % al 0.1 % con 3 minutos. Nótese el salto tan grande que se observa en todos los casos al pasar de $\alpha = 0$ a $\alpha = 1, 2, 3$, lo cual indica la importancia de la inclusión del margen en la capacidad máxima de las pilas.

En la Tabla 7.8 se presentan los resultados obtenidos para las instancias del conjunto *P33* con los algoritmos VND, GVNS y HVNS utilizando soluciones 2-factibles. Al compararse con la Tabla 7.5, con el mismo formato y donde se presentaban resultados análogos utilizando soluciones 0-factibles, se observa que se obtienen resultados notablemente mejores introduciendo 2 unidades extra en el tamaño de las pilas. En la VND la mejora no es

				10 seg		3 min	
Problema	LB	Best	VND	GVNS	HVNS	GVNS	HVNS
R00	911	1063	1.414	1.025	1.008	1.000	1.000
R01	875	1032	1.626	1.000	1.008	1.000	1.000
R02	935	1065	1.348	1.027	1.008	1.004	1.000
R03	961	1100	1.554	1.011	1.000	1.000	1.000
R04	937	1052	1.506	1.010	1.005	1.005	1.000
R05	900	1008	1.293	1.039	1.022	1.017	1.000
R06	998	1110	1.248	1.005	1.000	1.000	1.000
R07	963	1105	1.271	1.007	1.004	1.000	1.000
R08	978	1109	1.392	1.000	1.000	1.000	1.000
R09	976	1091	1.406	1.008	1.000	1.004	1.000
R10	901	1016	1.361	1.000	1.000	1.000	1.000
R11	892	1001	1.389	1.018	1.000	1.000	1.000
R12	984	1109	1.246	1.010	1.002	1.000	1.000
R13	956	1084	1.202	1.011	1.000	1.006	1.000
R14	879	1034	1.410	1.024	1.017	1.000	1.000
R15	985	1142	1.278	1.011	1.014	1.003	1.000
R16	967	1093	1.455	1.013	1.000	1.000	1.000
R17	946	1073	1.458	1.029	1.009	1.004	1.000
R18	1008	1118	1.392	1.038	1.028	1.007	1.007
R19	938	1089	1.275	1.005	1.006	1.005	1.002
Media			1.376	1.015	**1.006**	1.003	**1.001**

Tabla 7.8: VNS con soluciones 2-factibles (instancias 33 encargos)

muy grande debido a las propias limitaciones del algoritmo, pero en el caso del GVNS la desviación media baja de 12 % a 1.5 % en 10 segundos y de 8.3 % a 0.3 % en 3 minutos. De nuevo, el algoritmo con mejor comportamiento es el HVNS, específicamente adaptado al problema, con una desviación media del 0.6 % en sólo 10 segundos y obteniendo las mejores soluciones conocidas en 18 de las 20 instancias consideradas con 3 minutos de tiempo de cómputo. Nótese que con la introducción de soluciones 2-factibles en los algoritmos GVNS y HVNS se mejoran las soluciones de *todas* las instancias consideradas.

También es importante hacer notar que los resultados obtenidos con los algoritmos GVNS y HVNS utilizando soluciones 2-factibles mejoran sensiblemente los mejores resultados previos de la Sección 7.2, resumidos en la Tabla 7.1. Se obtienen soluciones mejores en *todas* las instancias consideradas tanto con 10 segundos como con 3 minutos de tiempo de cómputo, y con el nuevo algoritmo propuesto en este trabajo, el HVNS, la desviación media baja del 4 % al 0.6 % en 10 segundos y del 1 % al 0.1 % en 3 minutos.

7.5.2.3. Instancias con 66 encargos

En la gráfica de la Figura 7.5, que tiene el mismo formato que la Figura 7.4, se comparan los resultados obtenidos para las instancias del conjunto *P66* con los algoritmos GVNS y HVNS utilizando soluciones α-factibles para distintos valores de $\alpha = 0, 1, 2, 3$. Se observa un comportamiento similar al obtenido con las instancias del conjunto *P33*: la mayor desviación media siempre se obtiene con $\alpha = 0$, no utilizando ninguna unidad extra en el tamaño de las pilas de las soluciones intermedias, y no se observan diferencias significativas entre el uso de 1, 2 ó 3 unidades de tamaño extra. Esto reafirma de nuevo la importancia de las soluciones α-factibles, que permiten mejorar notablemente la eficiencia de ambos algoritmos en instancias de distinto tamaño.

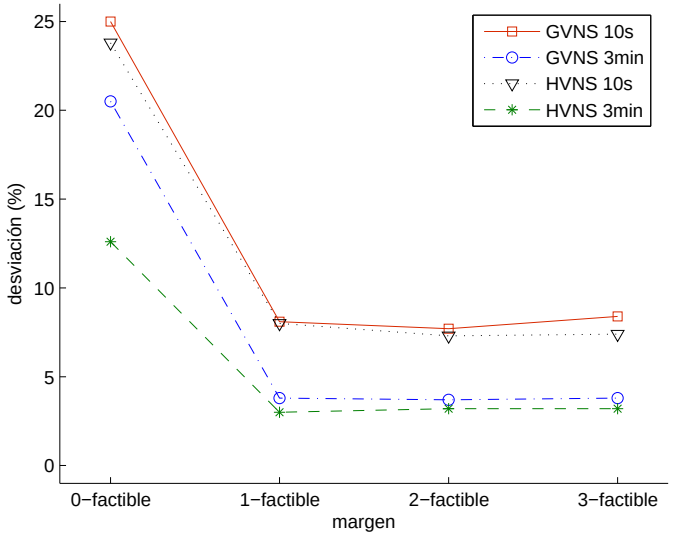


Figura 7.5: HVNS y GVNS con soluciones α-factibles, $\alpha = 0, 1, 2, 3$ (instancias 66 encargos)

En la Tabla 7.9 se presentan los resultados obtenidos para las instancias del conjunto *P66* con los algoritmos VND, GVNS y HVNS utilizando soluciones 2-factibles. De nuevo, en comparación con la Tabla 7.6, con el mismo formato y donde se presentaban resultados análogos utilizando soluciones 0-factibles, se observa que se obtienen resultados notablemente mejores introduciendo 2 unidades extra en el tamaño de las pilas, mejorándose las soluciones obtenidas con los algoritmos GVNS y HVNS en *todas* las instancias consideradas. Además, las desviaciones medias bajan significativamente, del 25 % al 7.7 % y del 20.5 % al 3.7 % en el GVNS y del 23.8 % al 7.3 % y del 12.6 % al 3.2 % en el HVNS.

En estas instancias del conjunto *P66* los resultados obtenidos con los algoritmos GVNS y HVNS utilizando soluciones 2-factibles también mejoran sensiblemente los mejores resultados previos, resumidos en la Tabla 7.2. De nuevo se consiguen mejorar las soluciones

				10 seg		3 min	
Problema	LB	Best	VND	GVNS	HVNS	GVNS	HVNS
R00-66	1237	1594	1.617	1.096	1.038	1.047	1.031
R01-66	1257	1600	1.298	1.093	1.064	1.049	1.043
R02-66	1295	1576	1.582	1.117	1.077	1.027	1.062
R03-66	1290	1630	1.610	1.102	1.060	1.034	1.026
R04-66	1295	1611	1.563	1.047	1.077	1.048	1.029
R05-66	1204	1534	1.327	1.110	1.065	1.028	1.029
R06-66	1294	1651	1.589	1.094	1.105	1.028	1.036
R07-66	1307	1653	1.525	1.050	1.064	1.040	1.010
R08-66	1297	1607	1.507	1.083	1.111	1.048	1.034
R09-66	1276	1598	1.419	1.074	1.086	1.023	1.031
R10-66	1339	1702	1.482	1.068	1.078	1.036	1.039
R11-66	1268	1575	1.315	1.046	1.067	1.039	1.053
R12-66	1295	1646	1.358	1.058	1.064	1.038	1.026
R13-66	1275	1616	1.561	1.066	1.088	1.025	1.026
R14-66	1245	1610	1.438	1.074	1.067	1.045	1.014
R15-66	1228	1604	1.433	1.080	1.067	1.044	1.022
R16-66	1356	1720	1.413	1.039	1.085	1.031	1.028
R17-66	1274	1627	1.598	1.101	1.075	1.053	1.051
R18-66	1328	1671	1.530	1.092	1.065	1.024	1.023
R19-66	1256	1635	1.314	1.060	1.052	1.034	1.029
Media			1.474	1.077	**1.073**	1.037	**1.032**

Tabla 7.9: VNS con soluciones 2-factibles (instancias 66 encargos)

de *todas* las instancias consideradas y la desviación media, en el caso del algoritmo HVNS, baja del 18 % al 7.3 % en 10 segundos y del 8 % al 3.2 % en 3 minutos.

7.5.2.4. Instancias con 132 encargos

En la gráfica de la Figura 7.6, que tiene el mismo formato que las Figuras 7.4 y 7.5, se comparan los resultados obtenidos para las instancias del conjunto *T132* con los algoritmos GVNS y HVNS utilizando soluciones α-factibles para distintos valores de $\alpha = 0, 1, 2, 3$. El comportamiento es el esperado, reafirmando de nuevo el impacto de las soluciones α-factibles: la mayor desviación se obtiene con $\alpha = 0$ y no hay diferencias significativas entre los valores de $\alpha = 1, 2, 3$.

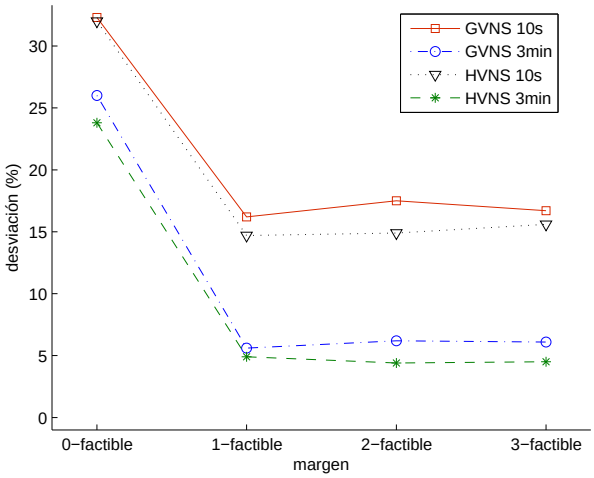


Figura 7.6: HVNS y GVNS con soluciones α-factibles, $\alpha = 0,1,2,3$ (instancias 132 encargos)

				10 seg		3 min	
Problema	LB	Best	VND	GVNS	HVNS	GVNS	HVNS
R00-132	1741	2591	1.431	1.222	1.134	1.068	1.069
R01-132	1789	2645	1.549	1.178	1.157	1.084	1.060
R02-132	1778	2639	1.584	1.180	1.142	1.071	1.019
R03-132	1807	2752	1.582	1.145	1.110	1.047	1.037
R04-132	1747	2603	1.516	1.153	1.100	1.053	1.040
R05-132	1753	2616	1.566	1.186	1.116	1.056	1.064
R06-132	1742	2576	1.345	1.163	1.093	1.063	1.054
R07-132	1712	2615	1.434	1.169	1.146	1.048	1.056
R08-132	1747	2638	1.437	1.185	1.165	1.077	1.063
R09-132	1708	2554	1.519	1.145	1.162	1.041	1.021
R10-132	1799	2646	1.532	1.197	1.187	1.068	1.060
R11-132	1779	2632	1.388	1.171	1.158	1.044	1.056
R12-132	1718	2555	1.466	1.175	1.179	1.086	1.056
R13-132	1765	2659	1.547	1.155	1.167	1.062	1.032
R14-132	1739	2605	1.404	1.149	1.126	1.050	1.016
R15-132	1729	2626	1.416	1.208	1.178	1.074	1.026
R16-132	1717	2534	1.410	1.197	1.172	1.058	1.036
R17-132	1694	2569	1.578	1.203	1.144	1.064	1.022
R18-132	1790	2652	1.391	1.168	1.154	1.053	1.031
R19-132	1764	2644	1.526	1.151	1.196	1.068	1.056
Media			1.481	1.175	**1.149**	1.062	**1.044**

Tabla 7.10: VNS con soluciones 2-factibles (instancias 132 encargos)

En la Tabla 7.10 se presentan los resultados obtenidos para las instancias del conjunto *T132* con los algoritmos VND, GVNS y HVNS utilizando soluciones 2-factibles. De nuevo se observa que se mejoran notablemente los resultados contenidos en la Tabla 7.7, con el mismo formato y donde se presentaban resultados análogos utilizando soluciones 0-factibles, mejorándose las soluciones obtenidas con los algoritmos GVNS y HVNS en *todas* las instancias consideradas. En este caso la mejora de las desviaciones medias es sensiblemente mayor que en las instancias de menor tamaño: de 32.3 % a 17.5 % y de 26.0 % a 6.2 % en el GVNS y del 32.0 % al 14.9 % y del 23.8 % al 4.4 % en el HVNS.

7.5.3. Mejores resultados obtenidos con VNS

En la Tabla 7.11 se presenta un resumen de los mejores resultados obtenidos con los algoritmos de Búsqueda en Entorno Variable para los conjuntos de instancias *P33*, *P66* y *T132*, todos ellos conseguidos con el nuevo algoritmo propuesto en este trabajo, el HVNS, y de los mejores resultados publicados en la literatura por otros autores para los conjuntos de instancias *P33*, *P66*, conseguidos con el algoritmo LNS. Los datos presentados representan la calidad media de las soluciones obtenidas con el tiempo de cómputo indicado para cada conjunto de instancias, distinguiendo los casos de $\alpha = 0$ y $\alpha > 0$ para el algoritmo HVNS. Nótese la mejora conseguida con el algoritmo HVNS ($\alpha > 0$) con respecto a los resultados de la LNS en las instancias de los conjuntos *P33* y *P66*.

	HVNS, $\alpha = 0$			HVNS, $\alpha > 0$			LNS	
	P33	*P66*	*T132*	*P33*	*P66*	*T132*	*P33*	*P66*
10 segundos	1.060	1.238	1.320	**1.006**	**1.073**	**1.149**	1.04	1.18
3 minutos	1.022	1.126	1.238	**1.001**	**1.032**	**1.044**	1.01	1.08

Tabla 7.11: Mejores resultados con VNS (HVNS) frente a mejores de la literatura (LNS)

7.5.4. Comparación de Estructuras de Entornos

En esta sección se presenta un estudio del impacto de cada estructura de entornos en el comportamiento de los algoritmos de Búsqueda en Entorno Variable y los resultados obtenidos por ellos. Intuitivamente, el impacto del uso de una estructura de entornos depende del tamaño de los entornos que genera y de la diversidad de las soluciones contenidas en ellos, pero en numerosas ocasiones es complicado determinar cómo afectan estos factores y dicho impacto no puede ser evaluado a priori. Para estudiar el comportamiento en la práctica de las distintas estructuras de entornos utilizadas se ha eliminado cada una de ellas por separado de la heurística HVNS y se han resuelto las instancias de los conjuntos *P33* y *P66* con las heurísticas HVNS modificadas.

En la Tabla 7.12 se muestran las diferencias entre la desviación media de las soluciones obtenidas con el algoritmo HVNS original y los algoritmos HVNS modificados. Que dicha diferencia sea positiva significa que la versión modificada (sin la estructura de entornos correspondiente) produce mejores resultados, en media, que la versión original (con todas las estructuras de entornos). También se presenta, entre paréntesis, el número de instancias de cada conjunto (sobre un total de 20) en las que la versión modificada produjo una solución mejor que la versión original. Las dos primeras columnas indican el tamaño de las instancias consideradas y el tiempo de cómputo utilizado, respectivamente, y las siguientes seis columnas contienen las diferencias entre las desviaciones medias de las versiones modificadas, sin cada estructura de entornos, y la original. En la penúltima línea se muestra el porcentaje medio de empeoramiento de las soluciones dadas por las versiones modificadas y en la última línea se presenta la proporción de instancias pertenecientes a todos los conjuntos considerados para las cuales las versiones modificadas produjeron mejores resultados medios que la heurística original.

Nótese que para comparar el comportamiento de las versiones modificadas con la versión original se fija el tiempo de cómputo disponible y el número de iteraciones a realizar se calcula en cada caso para no exceder dicho tiempo máximo. Así, la eliminación de una estructura de entornos produce un ahorro de tiempo de cómputo que será mayor cuanto mayores sean los entornos generados por dicha estructura, de manera que el número de iteraciones que puede realizar la versión modificada correspondiente crece y por tanto hay más tiempo disponible para explorar el resto de estructuras de entornos. Así, si la estructura de entornos considerada no ofrece soluciones buenas y diversas que ayuden lo suficiente en el proceso de búsqueda, al eliminarla es posible que se obtengan soluciones de mayor calidad al disponer de más tiempo de cómputo para el resto de estructuras.

Instancias	Tiempo	RS	CS	ISS	R	RP	SP
P33	10 seg	-0.3 (8)	4.2 (0)	0.7 (8)	9.1 (0)	0.4 (6)	0.0 (10)
P33	3 min	0.0 (7)	3.0 (0)	0.0 (4)	7.2 (0)	-0.1 (7)	0.2 (9)
P66	10 seg	0.6 (7)	6.7 (0)	2.4 (2)	24.1 (0)	1.1 (6)	0.8 (8)
P66	3 min	0.0 (12)	3.8 (2)	0.5 (12)	18.0 (0)	0.6 (8)	0.1 (8)
Dif. media (%)		0.07	**4.43**	0.90	**14.6**	0.50	0.27
# Mej. sol. modif.		$\frac{34}{80}$	$\frac{\mathbf{2}}{\mathbf{80}}$	$\frac{26}{80}$	$\frac{\mathbf{0}}{\mathbf{80}}$	$\frac{27}{80}$	$\frac{35}{80}$

Tabla 7.12: Impacto de las estructuras de entornos en el algoritmo HVNS

En la Tabla 7.12 se observa que hay sólo dos diferencias negativas, una para la estructura RS y otra para la RP, y la proporción de veces en la que la heurística modificada produce mejores soluciones que la original es siempre menor que un medio (y muy próxima a 0 en los casos de las estructuras CS y R). Esto justifica por sí mismo el uso de cada estructura de entornos y muestra que las que más impacto tienen en la calidad de las soluciones finales obtenidas son las estructuras CS y R. Nótese que eliminando la estructura de entornos

de Reinserción no se consiguen mejores soluciones en ninguno de los casos considerados y que el empeoramiento medio es mayor del 14 %, lo cual muestra el gran impacto de esta estructura de entornos en el comportamiento del algoritmo HVNS.

Los resultados de la Tabla 7.12 presentada anteriormente sirven para estimar el impacto en el comportamiento del algoritmo HVNS de cada estructura de entornos considerada *individualmente*. También resulta interesante estudiar el comportamiento del algoritmo HVNS cuando menos estructuras de entornos, por ejemplo sólo dos, son utilizadas. Las estructuras de entornos a utilizar deben ser elegidas adecuadamente, de manera que permitan la modificación de las rutas y las pilas de las soluciones durante el proceso de búsqueda. Consideraremos dos nuevas versiones del HVNS, una en la que sólo se utilizarán las estructuras de entornos introducidas en el trabajo de Petersen y Madsen (2009), RS y CS, y otra en la que se utilizarán las dos estructuras de entornos que mostraron tener un mayor impacto en el comportamiento del algoritmo HVNS al ser eliminadas, CS y R. Nótese que con la estructura RS se modifican las rutas de las soluciones, con la estructura CS se modifican las pilas y con la estructura R se modifican ambas.

La Tabla 7.13 tiene una estructura similar a la Tabla 7.12, presentándose ahora en la tercera columna los resultados obtenidos con el algoritmo HVNS utilizando sólo las estructuras de entornos RS y CS y en la cuarta las estructuras R y CS. Se observa que los resultados obtenidos eliminando todas las estructuras de entornos salvo RS y CS (columna RS&CS) son pobres, con un empeoramiento medio superior al 21 %, mientras que manteniendo sólo las estructuras R y CS (columna R&CS) los resultados son notablemente mejores pero aún son alrededor de un 2 % peores, en media, que los obtenidos con la versión en la que se utilizan todas las estructuras de entornos. Estos resultados también ratifican la importancia de utilizar todas las estructuras de entornos para aumentar la diversidad de las soluciones consideradas y flexibilizar el proceso de búsqueda.

Instancias	Tiempo	RS&CS	R&CS
P33	10s	14.9 (0)	1.4 (1)
P33	3min	10.2 (0)	0.1 (7)
P66	10s	35.3 (0)	4.5 (1)
P66	3min	27.2 (0)	2.0 (5)
Dif. media (%)		**21.90**	**2.00**
# Mej. sol. modif.		$\frac{\mathbf{0}}{\mathbf{80}}$	$\frac{\mathbf{14}}{\mathbf{80}}$

Tabla 7.13: Resultados de la HVNS con sólo 2 estructuras de entornos

7.6. Algoritmo Exterior

En la sección anterior se presentaron los resultados obtenidos con los algoritmos de Búsqueda en Entorno Variable, los cuales fueron mejorados notablemente con la introducción de soluciones intermedias que pudieran violar en algunas unidades la capacidad máxima de las pilas. Este hecho sugiere que permitiendo la violación temporal de otras restricciones del problema, como pueden ser las restricciones de precedencia, se podrían mejorar los resultados más aún, y así surgen lo que denominamos soluciones α-potenciales y el algoritmo Exterior, que se encarga de manejarlas y guiar el proceso de búsqueda a través de estas soluciones.

P33	10s	3min	P66	10s	3min	T132	10s	3min
R00	1.047	1.005	R00-66	1.167	1.120	R00-132	1.247	1.192
R01	1.016	1.004	R01-66	1.174	1.133	R01-132	1.228	1.205
R02	1.034	1.012	R02-66	1.168	1.138	R02-132	1.281	1.214
R03	1.052	1.015	R03-66	1.151	1.096	R03-132	1.212	1.189
R04	1.045	1.020	R04-66	1.117	1.125	R04-132	1.239	1.189
R05	1.026	1.020	R05-66	1.154	1.100	R05-132	1.228	1.182
R06	1.053	1.015	R06-66	1.125	1.104	R06-132	1.252	1.191
R07	1.041	1.007	R07-66	1.163	1.109	R07-132	1.240	1.182
R08	1.041	1.021	R08-66	1.146	1.119	R08-132	1.223	1.182
R09	1.048	1.018	R09-66	1.144	1.110	R09-132	1.229	1.188
R10	1.037	1.002	R10-66	1.176	1.128	R10-132	1.269	1.211
R11	1.038	1.010	R11-66	1.151	1.119	R11-132	1.233	1.180
R12	1.043	1.020	R12-66	1.139	1.103	R12-132	1.261	1.205
R13	1.030	1.015	R13-66	1.180	1.136	R13-132	1.237	1.203
R14	1.045	1.014	R14-66	1.185	1.140	R14-132	1.210	1.184
R15	1.039	1.027	R15-66	1.167	1.118	R15-132	1.240	1.198
R16	1.032	1.006	R16-66	1.165	1.124	R16-132	1.265	1.219
R17	1.042	1.019	R17-66	1.185	1.141	R17-132	1.233	1.195
R18	1.064	1.034	R18-66	1.212	1.138	R18-132	1.215	1.187
R19	1.034	1.017	R19-66	1.175	1.141	R19-132	1.219	1.183
	1.040	1.015		1.162	1.122		1.238	1.194

Tabla 7.14: Resultados del algoritmo Exterior (instancias 33, 66 y 132 encargos)

En esta sección se presentan los resultados obtenidos con el algoritmo Exterior (EXT, Sección 6.3), que utiliza las estructuras de entornos introducidas en la Sección 4.4 para incluir soluciones α-potenciales en el proceso de búsqueda. En la Tabla 7.14 se muestra la calidad de las soluciones obtenidas por el algoritmo EXT para las instancias de los conjuntos *P33* (primeras 3 columnas), *P66* (columnas 4 a 6) y *T132* (últimas 3 columnas), utilizando un tiempo de cómputo de 10 segundos y 3 minutos. En la primera columna de cada terna se indica el nombre de las instancias consideradas y en la última fila la calidad

media de las soluciones obtenidas en cada caso.

Se observa que los resultados obtenidos con el algoritmo EXT son significativamente peores para todos los conjuntos de instancias que los obtenidos anteriormente con el algoritmo HVNS (Tablas 7.8-7.10), pero son comparables con los mejores resultados previos de la Sección 7.2 (algoritmo LNS, Tablas 7.1 y 7.2): para las instancias con 33 encargos se tienen desviaciones medias del 4 % en ambos algoritmos con 10 segundos de tiempo de cómputo y del 1 % con LNS frente al 1.5 % con EXT con 3 minutos; para las instancias con 66 encargos se tienen desviaciones medias del 18 % con LNS frente al 16.2 % con EXT en 10 segundos y del 8 % con LNS frente al 12.2 % con EXT en 3 minutos.

La gráfica de la Figura 7.7 presenta una comparación directa de los resultados obtenidos para los conjuntos de instancias *P33*, *P66* y *T132* (representados en el eje OX) con el algoritmo EXT y el algoritmo HVNS, que fue el algoritmo de Búsqueda en Entorno Variable con mejor comportamiento global. Se observa claramente que el algoritmo HVNS mejora al EXT en todos los conjuntos de instancias, ofreciendo incluso mejores resultados medios en 10 segundos que el algoritmo EXT en 3 minutos.

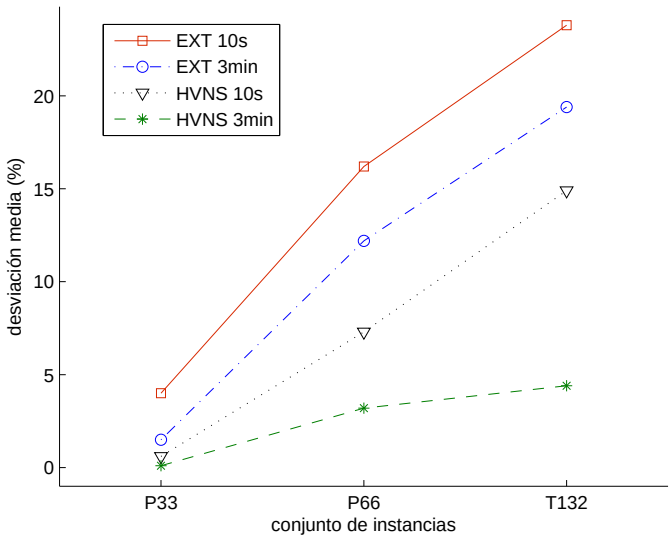


Figura 7.7: Comparación de algoritmos HVNS y EXT (instancias 33, 66 y 132 encargos)

7.7. Búsqueda en Entorno Variable Híbrida Exterior

En la sección anterior se comprueba que el algoritmo HVNS tiene un comportamiento mejor que el algoritmo EXT en todos los conjuntos de instancias considerados, lo que lleva a concluir que la Búsqueda Exterior, por sí sola, no es tan eficaz como la Búsqueda en

Entorno Variable. Sin embargo, esto no quiere decir que no pueda resultar de utilidad en combinación con otras heurísticas. En esta sección se presentarán los resultados obtenidos con el algoritmo de Búsqueda en Entorno Variable Híbrida Exterior (EHVNS, Sección 6.3), con los que se comprobará que la Búsqueda en Entorno Variable Híbrida mejora su comportamiento al combinarse adecuadamente con el proceso de Búsqueda Exterior.

En la Tabla 7.15 se muestra, con el mismo formato que la Tabla 7.14, la calidad de las soluciones obtenidas por el algoritmo EHVNS para las instancias de los conjuntos *P33*, *P66* y *T132*, y en la Figura 7.8 se presenta, con el mismo formato que la figura 7.7, una comparación directa de dichos resultados con los obtenidos por el algoritmo HVNS sin Búsqueda Exterior. En ambas se observa que para las instancias de *P33* los resultados de ambos algoritmos son prácticamente idénticos, mientras que para las instancias de *P66* y *T132* los resultados medios del algoritmo EHVNS son mejores que los del algoritmo HVNS sin Búsqueda Exterior: la desviación media baja del 7.3 % al 6.8 % (10 segundos) y del 3.2 % al 2.5 % (3 minutos) en las instancias con 66 encargos y del 14.9 % al 11.1 % (10 segundos) y del 4.4 % al 4.1 % (3 minutos) en las instancias con 132 encargos.

P33	10s	3min	P66	10s	3min	T132	10s	3min
R00	1.000	1.000	R00-66	1.078	1.018	R00-132	1.110	1.058
R01	1.008	1.000	R01-66	1.078	1.022	R01-132	1.110	1.035
R02	1.011	1.000	R02-66	1.082	1.069	R02-132	1.130	1.068
R03	1.000	1.000	R03-66	1.060	1.020	R03-132	1.096	1.021
R04	1.019	1.000	R04-66	1.096	1.017	R04-132	1.088	1.063
R05	1.003	1.000	R05-66	1.067	1.007	R05-132	1.099	1.018
R06	1.000	1.000	R06-66	1.075	1.041	R06-132	1.133	1.064
R07	1.004	1.000	R07-66	1.058	1.007	R07-132	1.106	1.031
R08	1.029	1.000	R08-66	1.066	1.023	R08-132	1.114	1.022
R09	1.005	1.000	R09-66	1.059	1.023	R09-132	1.116	1.026
R10	1.000	1.000	R10-66	1.056	1.008	R10-132	1.138	1.062
R11	1.000	1.000	R11-66	1.057	1.005	R11-132	1.108	1.019
R12	1.016	1.000	R12-66	1.083	1.012	R12-132	1.135	1.052
R13	1.006	1.000	R13-66	1.046	1.025	R13-132	1.078	1.021
R14	1.000	1.000	R14-66	1.033	1.021	R14-132	1.112	1.051
R15	1.011	1.000	R15-66	1.066	1.026	R15-132	1.094	1.043
R16	1.000	1.000	R16-66	1.063	1.028	R16-132	1.131	1.064
R17	1.004	1.000	R17-66	1.066	1.053	R17-132	1.131	1.053
R18	1.021	1.007	R18-66	1.070	1.029	R18-132	1.118	1.023
R19	1.013	1.002	R19-66	1.094	1.043	R19-132	1.075	1.036
	1.007	1.001		**1.068**	**1.025**		**1.111**	**1.041**

Tabla 7.15: Resultados de la EHVNS (instancias 33, 66 y 132 encargos)

Nótese la diferencia entre las gráficas de las Figuras 7.7 y 7.8: en la primera se observa que las dos líneas que representan los resultados obtenidos con el algoritmo EXT están claramente por encima de las asociadas al algoritmo HVNS, mostrando la superioridad de este último sobre el algoritmo EXT; sin embargo, en la segunda la situación se invierte y, para cada tiempo de cómputo, la línea del algoritmo EHVNS pasa a estar por debajo de la asociada al HVNS, mostrando la mejoría conseguida al combinar las dos técnicas más relevantes introducidas en este trabajo, la Búsqueda en Entorno Variable Híbrida y la Búsqueda Exterior. Los resultados anteriores permiten concluir que, efectivamente, el algoritmo con mejor comportamiento global en todas las instancias consideradas es el algoritmo combinado, EHVNS.

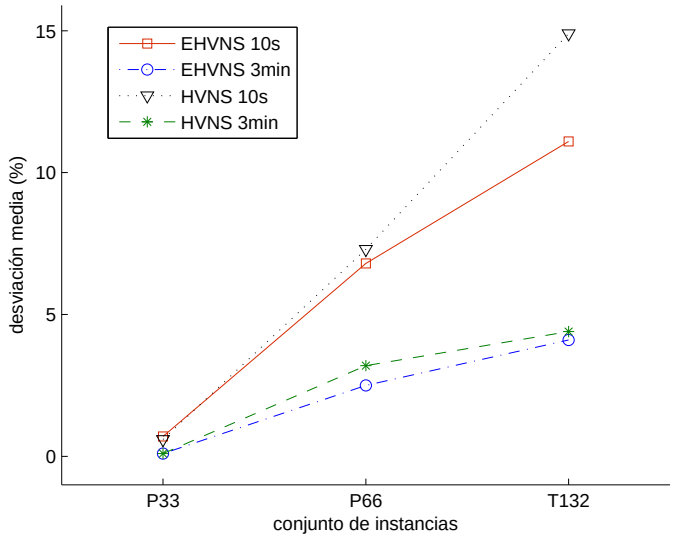


Figura 7.8: Comparación de algoritmos HVNS y EHVNS (instancias 33, 66 y 132 encargos)

7.8. Comparativa con resultados previos

En esta sección se hace una comparación exhaustiva de los mejores resultados previos publicados en la literatura por otros autores para todas las instancias consideradas en este capítulo, obtenidos con el algoritmo LNS (Sección 7.2), con los resultados obtenidos con el algoritmo propuesto en este trabajo que mejor comportamiento general ha mostrado, la Búsqueda en Entorno Variable Híbrida Exterior.

Para las instancias del conjunto *P12*, con sólo 12 encargos, los algoritmos LNS y EHVNS tienen un comportamiento similar (al igual que otros algoritmos como GVNS y HVNS), obteniéndose soluciones óptimas para todas ellas con un tiempo máximo de cómputo 10 segundos por instancia.

En la Tabla 7.16 se presenta la calidad de las soluciones obtenidas con los algoritmos LNS y EHVNS para las instancias del conjunto *P33*, obtenidas con 10 segundos (columnas cuarta y quinta) y 3 minutos (dos últimas columnas) de tiempo de cómputo. En la segunda columna se incluye una cota inferior para cada instancia y en la tercera la mejor solución conocida, obtenida con el algoritmo LNS (Petersen y Madsen, 2009). En dicha tabla se observa que con el algoritmo propuesto en este trabajo, el EHVNS, se obtienen soluciones sensiblemente mejores en todas las instancias con 33 encargos, consiguiendo una desviación media menor del 1 % en sólo 10 segundos y obteniendo las mejores soluciones conocidas en 18 de las 20 instancias en sólo 3 minutos.

			10 sec		3 min	
Instancia	LB	Best	LNS	EHVNS	LNS	EHVNS
R00	911	1063	1.04	1.000	1.01	1.000
R01	875	1032	1.04	1.008	1.01	1.000
R02	935	1065	1.04	1.011	1.01	1.000
R03	961	1100	1.06	1.000	1.01	1.000
R04	937	1052	1.05	1.019	1.02	1.000
R05	900	1008	1.03	1.003	1.01	1.000
R06	998	1110	1.06	1.000	1.02	1.000
R07	963	1105	1.05	1.004	1.01	1.000
R08	978	1109	1.04	1.029	1.01	1.000
R09	976	1091	1.04	1.005	1.01	1.000
R10	901	1016	1.05	1.000	1.00	1.000
R11	892	1001	1.06	1.000	1.01	1.000
R12	984	1109	1.04	1.016	1.01	1.000
R13	956	1084	1.04	1.006	1.01	1.000
R14	879	1034	1.03	1.000	1.00	1.000
R15	985	1142	1.04	1.011	1.01	1.000
R16	967	1093	1.02	1.000	1.00	1.000
R17	946	1073	1.04	1.004	1.00	1.000
R18	1008	1118	1.05	1.021	1.01	1.007
R19	938	1089	1.03	1.013	1.01	1.002
Media			1.04	1.007	1.01	1.001

Tabla 7.16: Comparación de los mejores resultados (instancias 33 encargos)

Ejecutando el algoritmo EHVNS con un tiempo de cómputo grande se obtienen exactamente las mismas mejores soluciones obtenidas con el algoritmo LNS para las 20 instancias con 33 encargos, lo que sugiere que, probablemente, muchas de ellas son óptimas.

En la Tabla 7.17 se presenta la calidad de las soluciones obtenidas con los algoritmos LNS y EHVNS para las instancias del conjunto *P66* siguiendo el mismo formato que la Tabla 7.16, con la única salvedad de la inclusión de una última columna en la que se

presentan los costes de las mejores soluciones obtenidas con el algoritmo EHVNS. En dicha tabla se observa que, de nuevo, el algoritmo que combina la Búsqueda en Entorno Variable con la Búsqueda Exterior, el EHVNS, produce soluciones notablemente mejores en todas las instancias con 66 encargos, disminuyendo la desviación media del 18 % al 6.8 % en 10 segundos y del 8 % al 2.5 % en 3 minutos. Nótese que se obtienen mejores soluciones en todas las instancias con 66 encargos, e incluso las soluciones obtenidas por el algoritmo EHVNS en 10 segundos son, en media, mejores que las obtenidas con el algoritmo LNS en 3 minutos.

			10 sec		3 min		
Instancia	LB	Best	LNS	EHVNS	LNS	EHVNS	Best*
R00-66	1237	1594	1.19	1.078	1.07	1.018	1597
R01-66	1257	1600	1.20	1.078	1.08	1.022	1600
R02-66	1295	1576	1.20	1.082	1.12	1.069	1576
R03-66	1290	1631	1.14	1.060	1.06	1.020	**1630**
R04-66	1295	1611	1.18	1.096	1.09	1.017	1626
R05-66	1204	1528	1.18	1.067	1.07	1.007	**1524**
R06-66	1294	1651	1.17	1.075	1.07	1.041	**1644**
R07-66	1307	1653	1.17	1.058	1.08	1.007	1655
R08-66	1297	1607	1.18	1.066	1.07	1.023	1612
R09-66	1276	1598	1.18	1.059	1.08	1.023	**1593**
R10-66	1339	1702	1.17	1.056	1.09	1.008	1702
R11-66	1268	1575	1.19	1.057	1.08	1.005	1575
R12-66	1295	1652	1.19	1.083	1.10	1.012	**1646**
R13-66	1275	1617	1.19	1.046	1.10	1.025	**1616**
R14-66	1245	1611	1.21	1.033	1.09	1.021	**1608**
R15-66	1228	1608	1.19	1.066	1.10	1.026	**1604**
R16-66	1356	1725	1.16	1.063	1.07	1.028	**1720**
R17-66	1274	1627	1.21	1.066	1.10	1.053	1627
R18-66	1328	1671	1.18	1.070	1.08	1.029	1676
R19-66	1256	1635	1.17	1.094	1.09	1.043	1635
Media			1.18	1.068	1.08	1.025	

Tabla 7.17: Comparación de los mejores resultados (instancias 66 encargos)

En la última columna de la Tabla 7.17 se presentan en negrita los costes de las mejores soluciones obtenidas con EHVNS que mejoran las mejores soluciones conocidas hasta el momento (obtenidas con LNS) y subrayados los costes de las que igualan las mejores soluciones conocidas hasta el momento. Se observa que con el algoritmo EHVNS se consiguen mejorar las mejores soluciones conocidas de 9 de las instancias y se igualan las de otras 6, obteniéndose soluciones peores en sólo las 5 instancias restantes.

No existen resultados previos publicados por otros autores en la literatura sobre ins-

tancias con más de 66 encargos, por lo que no es posible la comparación con los mejores resultados obtenidos con el algoritmo EHVNS para las instancias del conjunto *T132*. Alternativamente, en la Tabla 7.18 se comparan las mejores soluciones encontradas con el algoritmo HVNS (fila etiquetada como *Best*) y las mejores encontradas al combinarlo con la Búsqueda Exterior (fila etiquetada como *Best**), comprobando que con el algoritmo EHVNS se consiguen mejorar todas ellas, dando lugar a las mejores soluciones conocidas para estas instancias. Para simplificar la notación de la Tabla 7.18 se ha suprimido la última parte de los nombres de las instancias, "-132", que indica que las instancias tienen 132 encargos.

	R00	R01	R02	R03	R04	R05	R06	R07	R08	R09
Best	2591	2645	2639	2752	2603	2616	2576	2615	2638	2554
Best*	2555	2644	2620	2724	2538	2601	2572	2550	2601	2492
	R10	R11	R12	R13	R14	R15	R16	R17	R18	R19
Best	2646	2632	2555	2659	2605	2626	2534	2569	2652	2644
Best*	2576	2593	2548	2639	2572	2578	2524	2486	2594	2604

Tabla 7.18: Mejores soluciones obtenidas por HVNS y EHVNS (instancias 132 encargos)

7.9. Comparativa Temple Simulado

En las secciones precedentes se han presentado los resultados obtenidos con los algoritmos de Búsqueda en Entorno Variable (VND, GVNS, HVNS), el algoritmo Exterior (EXT) y el algoritmo combinado EHVNS, en los que se utilizan varias estructuras de entornos que comprenden tanto soluciones factibles como α-potenciales. También sería interesante comprobar si alguna otra heurística con una filosofía distinta a estas, en la cual sólo se utilice una estructura de entornos para guiar el proceso de búsqueda, podría dar lugar a soluciones de buena calidad.

En Petersen y Madsen (2009) se propone un algoritmo de Temple Simulado en el que la búsqueda local se realiza mediante las estructuras de entornos RS y CS; este algoritmo produce mejores resultados que otras heurísticas propuestas en dicho trabajo, como son la Búsqueda Tabú y la Búsqueda Local Iterativa. Sin embargo, el estudio presentado en la Sección 7.5.4 muestra que el entorno con mayor impacto sobre la calidad de las soluciones finales es el de Reinserción. Esto motiva el contenido de esta sección, en la cual compararemos la implementación del algoritmo de Temple Simulado (SA) de Petersen y Madsen (2009) con una nueva implementación de dicho algoritmo en la que la estructura de entornos de Reinserción es la encargada de guiar la búsqueda local (SAR, introducido en la Sección 6.2). Para aprovechar toda la potencia de esta estructura es necesaria la introducción de soluciones intermedias α-factibles y el uso de los algoritmos de reducción

de pilas introducidos en la Sección 5.1.2.

En la Tabla 7.19 se compara la calidad de las soluciones obtenidas con los algoritmos SA y SAR para las instancias de los conjuntos *P33* y *P66*. La primera columna contiene los nombres de las instancias del conjunto *P33* (33 encargos), de manera que los nombres de las correspondientes instancias del conjunto *P66* (66 encargos) se obtienen simplemente añadiendo "-66" a los nombres anteriores. Las columnas de la segunda a la quinta muestran la calidad de las soluciones obtenidas para las instancias de *P33* y las cuatro siguientes para las instancias de *P66*, utilizando un tiempo de cómputo de 10 segundos y 3 minutos.

	P33				*P66*			
Instancia	10 seg		3 min		10 seg		3 min	
	SA	SAR	SA	SAR	SA	SAR	SA	SAR
R00	1.26	1.000	1.13	1.026	1.38	1.074	1.21	1.031
R01	1.17	1.037	1.08	1.008	1.39	1.090	1.22	1.061
R02	1.19	1.046	1.10	1.004	1.38	1.131	1.28	1.074
R03	1.22	1.056	1.12	1.000	1.35	1.086	1.23	1.036
R04	1.25	1.065	1.11	1.010	1.46	1.050	1.28	1.024
R05	1.22	1.049	1.15	1.020	1.40	1.068	1.21	1.022
R06	1.19	1.046	1.11	1.005	1.38	1.105	1.21	1.042
R07	1.23	1.055	1.11	1.016	1.44	1.088	1.22	1.016
R08	1.21	1.072	1.11	1.021	1.41	1.080	1.27	1.038
R09	1.15	1.020	1.08	1.005	1.35	1.040	1.25	1.035
R10	1.24	1.077	1.15	1.003	1.35	1.093	1.23	1.026
R11	1.24	1.024	1.10	1.027	1.40	1.130	1.25	1.030
R12	1.21	1.008	1.13	1.007	1.38	1.086	1.24	1.048
R13	1.23	1.061	1.08	1.007	1.39	1.072	1.24	1.037
R14	1.22	1.035	1.11	1.000	1.38	1.079	1.24	1.043
R15	1.21	1.011	1.10	1.000	1.38	1.106	1.22	1.032
R16	1.20	1.023	1.10	1.000	1.34	1.075	1.17	1.038
R17	1.24	1.021	1.12	1.004	1.40	1.100	1.29	1.052
R18	1.20	1.057	1.13	1.019	1.40	1.095	1.23	1.041
R19	1.16	1.059	1.11	1.006	1.41	1.099	1.22	1.040
Media	1.21	1.041	1.11	1.009	1.39	1.087	1.24	1.038

Tabla 7.19: Comparación entre SA y SAR

La Tabla 7.19 muestra claramente que la versión del Temple Simulado con la estructura de entornos R (SAR) produce mejores resultados que la versión con las estructuras RS y CS (SA). Para las instancias de tamaño 33 la desviación media baja del 21 % al 4.1 % con 10 segundos de tiempo de cómputo y del 11 % a menos del 1 % con 3 minutos, mientras que para las instancias de tamaño 66 baja del 39 % al 8.7 % con 10 segundos y del 24 % a menos del 4 % con 3 minutos. Estos resultados muestran de nuevo la potencia del operador

de Reinserción y enfatizan su capacidad para modificar al mismo tiempo las rutas y las pilas de las soluciones del DTSPMS, ya que puede utilizarse de forma autónoma sin la ayuda de ningún otro operador.

Capítulo 8

Conclusiones y aportaciones. Investigación futura

8.1. Conclusiones

Destacamos las siguientes conclusiones extraídas de esta monografía:

Descripción del problema

- Se ha descrito con detalle un problema de rutas de vehículos con restricciones adicionales de precedencia y carga, denominado el Doble Problema del Viajante con Múltiples Pilas (DTSPMS), que ha sido poco tratado por el momento en la literatura especializada, y se ha relacionado dicho problema con otros ya abordados en la literatura.

Cálculo de cotas inferiores

- Se ha estudiado la complejidad del DTSPMS y se ha propuesto un modelo de programación matemática alternativo que facilita la aplicación de técnicas de Relajación Lagrangiana y de Generación de Columnas (para la resolución de la descomposición de Dantzig-Wolfe) para la obtención de cotas inferiores del problema. Sin embargo, dichas técnicas han sido probadas en distintas instancias y no han proporcionado resultados satisfactorios.
- Se ha desarrollado un algoritmo de generación de planos de corte para la obtención de cotas inferiores que se aplica sobre una relajación del DTSPMS en la que se eliminan

las restricciones de precedencia. Con dicho algoritmo se han mejorado algunas cotas inferiores de instancias con 12 encargos y 2 pilas y 18 encargos y 3 pilas que no han podido ser resueltas de forma exacta en la literatura.

Descripción de las soluciones

- Se ha introducido una representación nueva para las soluciones factibles del DTSPMS en función de asignaciones y permutaciones que es más adecuada para el diseño de estructuras de entornos y algoritmos heurísticos que la representación con variables binarias. Además se ha ampliado el espacio de soluciones consideradas y se han introducido las soluciones denominadas α-factibles y potenciales, que son soluciones que incumplen algunas restricciones del problema y que han resultado útiles para flexibilizar, guiar y mejorar el proceso de búsqueda.
- Se han diseñado varios métodos para la generación de soluciones iniciales variadas y de buena calidad para el DTSPMS. Se han considerado métodos para la generación de distintos tipos de soluciones: soluciones factibles, soluciones α-factibles y soluciones α-potenciales.

Estructuras de entornos

- Se han desarrollado nuevas estructuras de entornos adaptadas al DTSPMS que permiten explotar la estructura de las soluciones del problema y guiar los procesos de búsqueda local en base a distintos criterios. Dichas estructuras de entornos se describen con rigor, proporcionando una descripción matemática precisa de todas ellas, estudiando el tamaño de los entornos que se forman a partir de cada estructura y haciendo especial hincapié en la descripción de los movimientos a realizar para obtener soluciones vecinas y mantener la factibilidad.
- Las nuevas estructuras de entornos para soluciones factibles se han extendido a soluciones α-factibles y se han desarrollado otras estructuras nuevas para soluciones potenciales.

Tratamiento de la infactibilidad

- Se ha propuesto la introducción de dos tipos distintos de infactibilidades en el DTSPMS, la Infactibilidad por Capacidades (IC) y la Infactibilidad por Precedencias (IP), con las que se ha conseguido un proceso de búsqueda más flexible que permite acceder a zonas del espacio de soluciones antes no accesibles.

- El manejo de las infactibilidades IC e IP, representadas por las soluciones α-factibles y α-potenciales, ha sido tratado con detalle y rigor, además de ilustrar los nuevos conceptos introducidos para ello con una amplia colección de ejemplos. Para eliminar la infactibilidad IC sin aumentar el coste de las soluciones se han desarrollado varios algoritmos, estrechamente relacionados entre sí, para los cuales se ha dado una descripción detallada y se ha estudiado su complejidad. Para eliminar la infactibilidad IP, que es un proceso más complejo, se han propuesto también varios operadores de proyección, que en este caso sí pueden alterar el coste de las soluciones modificadas.
- Se han propuesto distintas medidas para evaluar la infactibilidad de las soluciones consideradas, tanto IC como IP, que han servido para controlar y dirigir el proceso de búsqueda exterior de forma adecuada.
- Se ha analizado con detalle la relación que existe entre las infactibilidades IC e IP y se ha determinado que la infactibilidad IC siempre puede transformarse en infactibilidad IP pero que el recíproco no es cierto. Además, se ha estudiado cómo ambas infactibilidades pueden combinarse en un proceso de búsqueda conjunto con soluciones α-potenciales.

Diseño de heurísticas

- Se han desarrollado algoritmos heurísticos basados en distintas filosofías (Búsqueda en Entorno Variable, Temple Simulado, Búsqueda Exterior) que han proporcionado soluciones de buena calidad para instancias de tamaño realista del DTSPMS utilizando poco tiempo de cómputo. Entre ellos cabe destacar el algoritmo de Búsqueda en Entorno Variable Híbrida, el algoritmo Exterior y la Búsqueda en Entorno Variable Híbrida Exterior.

Experiencia computacional

- Se ha realizado una extensa experiencia computacional con instancias de distintos tamaños para evaluar el comportamiento de los algoritmos utilizados para la resolución del DTSPMS y se han relacionado y comentado con detalle los resultados obtenidos.
- Se han generado nuevas instancias con 132 encargos para evaluar el comportamiento de los algoritmos cuando se aplican a ejemplos test de tamaño mayor que los existentes en la literatura. La calibración de parámetros de las heurísticas se ha realizado sobre instancias de diferentes tamaños generadas a tal efecto.

Comparación de resultados

- Se han comparado los resultados obtenidos por los algoritmos de Búsqueda en Entorno Variable con y sin la utilización de soluciones α-factibles y se ha observado que la introducción de dichas soluciones mejora su eficiencia. Se ha constatado que el algoritmo de Búsqueda en Entorno Variable que se comporta mejor en general es la Búsqueda en Entorno Variable Híbrida, específicamente diseñada para el DTSPMS, con la cual se mejoran significativamente los resultados previos publicados en la literatura por otros autores.
- Se ha observado que la Búsqueda en Entorno Variable Híbrida produce mejores resultados que el Algoritmo Exterior, pero el algoritmo que muestra un mejor comportamiento global es el que combina ambas técnicas, la Búsqueda en Entorno Variable Híbrida Exterior. Se consiguen soluciones de buena calidad para todas las instancias de distintos tamaños consideradas con poco tiempo de cómputo, mejorando los mejores resultados publicados en la literatura por otros autores, y se encuentran nuevas mejores soluciones en varias de las instancias consideradas.

8.2. Principales aportaciones

Los resultados principales de la tesis han sido presentados en:

- EURO XXII. Praga (República Checa). Julio 2007.
- XXX Congreso Nacional de Estadística e Investigación Operativa. Valladolid (España). Septiembre 2007.
- ECCO XXI. Dubrovnik (Croacia). Mayo 2008.
- IWOR 2008. Móstoles (España). Junio 2008.
- Transnova 2008. Santa Cruz de Tenerife (España). Junio 2008.
- FLINS 2008. Madrid (España). Septiembre 2008.
- XXXI Congreso Nacional de Estadística e Investigación Operativa. Murcia (España). Febrero 2009.
- ECCO XXII. Jerusalén (Israel). Mayo 2009.

Además, se han realizado las siguientes publicaciones:

- A. Felipe, M.T. Ortuño y G. Tirado (2008) Neighborhood structures to solve the double traveling salesman problem (TSP) with multiple stacks using local search. *World Scientific Proceedings Series on Computer Engineering and Information Science I*, 701-706.
- A. Felipe, M.T. Ortuño y G. Tirado (2009) New neighborhood structures for the Double Traveling Salesman Problem With Multiple Stacks. *TOP* 17, 190-213.
- A. Felipe, M.T. Ortuño y G. Tirado (2009) The Double Traveling Salesman Problem With Multiple Stacks: A Variable Neighborhood Search Approach. *Computers and Operations Research* 36, 2983-2993.

8.3. Líneas futuras de investigación

A continuación se proponen algunas ideas sobre la resolución del DTSPMS que pueden ser prometedoras para establecer líneas futuras de investigación:

- **Generación de instancias con solución óptima conocida.** Al evaluar la calidad de las soluciones obtenidas por los algoritmos heurísticos para instancias de tamaño realista (33, 66 encargos) las soluciones que se toman como referencia son las mejores soluciones conocidas, ya que no se dispone de las soluciones óptimas. Sería interesante desarrollar un generador de instancias test con solución óptima conocida para poder utilizar dichas instancias para la evaluación de las heurísticas y comparar directamente con las soluciones óptimas.
- **Generación de instancias con distintas características.** Generar instancias en las que las localizaciones de recogida y entrega no estén uniformemente distribuidas en el plano sino que formen clústers, haya localizaciones aisladas, el depósito esté en una posición céntrica o desplazada, etc. Aplicar las heurísticas propuestas a diversos tipos de instancias permitiría evaluar qué algoritmos funcionan mejor para cada tipo y podría ofrecer indicaciones de cómo mejorar dichos algoritmos.
- **Algoritmos constructivos:** Desarrollar nuevas técnicas para la construcción de soluciones factibles de calidad.
- **Subproblemas del DTSPMS.** Estudiar con detalle la complejidad de algunos subproblemas del DTSPMS, manteniendo el límite de capacidad en las pilas, y plantear su aplicación a la resolución del problema original. Un subproblema interesante que ya ha sido utilizado en la resolución del problema original es el de determinar si una

pareja de rutas de recogida y entrega admite una asignación de pilas factible con m pilas de capacidad Q.

- **Enfoques exactos.** Explorar modelizaciones y relajaciones alternativas del problema y desarrollar algoritmos de branch and bound o branch and cut que sean capaces de resolver instancias del DTSPMS con más de 20 encargos.
- **Heurísticas.** Aplicar otras heurísticas con una filosofía diferente a la resolución de instancias de tamaño medio: algoritmos genéticos, algoritmos de colonia de hormigas, búsqueda dispersa, etc.
- **Casos particulares.** Adaptar los algoritmos que ya existen a casos particulares en los cuales una de las regiones posee menos localizaciones que la otra y diseñar otros nuevos que aprovechen la nueva estructura del problema para resolverlo de forma más eficiente.
- **Extensiones del problema.** Abordar la resolución de las extensiones más complejas del DTSPMS que puedan ajustarse mejor a las aplicaciones reales: flotas con varios vehículos, ventanas de tiempo en las recogidas y entregas de productos, restricciones de carga en dos o tres dimensiones, encargos de productos con diferentes formas y tamaños, recogidas y entregas simultáneas, etc.

Conclusions

The most relevant conclusions following from the thesis are presented next.

Problem description

- A vehicle routing problem with precedence and loading constraints called the Double Traveling Salesman Problem with Multiple Stacks (DTSPMS) has been described and formulated in detail, and some other related problems from the literature have been introduced.

Calculation of lower bounds

- The complexity of the DTSPMS has been studied and an alternative mathematical programming formulation has been proposed. Lagrangian relaxation and column generation techniques have been applied to this new formulation to obtain lower bounds for the problem, but the results obtained for the considered instances were not satisfactory.
- A cutting plane method to be applied to the relaxation of the problem obtained eliminating the precedence constraints has also been developed. With this approach some lower bounds for instances with 12 orders and 2 available stacks and 18 orders and 3 available stacks, that could not be solved to optimality in the literature, have been improved.

Solutions description

- A new representation of the feasible solutions of the DTSPMS in terms of assignments and permutations has been proposed and it has been used in the definition of neighborhood structures and moves to construct solutions. Two new kinds of solutions have been introduced: solutions with extra capacity (violating capacity constraints)

and solutions with conflicts (violating precedence constraints). These non-feasible solutions have diversified and improved the search process and made it more flexible.

- Different algorithms to generate initial varied good quality solutions have been developed, concerning feasible solutions, solutions with extra capacity and solutions with conflicts.

Neighborhood structures

- New neighborhood structures adapted to the DTSPMS and exploiting the structures of the solutions of the problem have been designed. They have been fully described: a detailed mathematical description of each of them has been provided and the implementation of feasible moves to create neighbors have been emphasized.
- These neighborhood structures have been extended to solutions with extra capacity and two new ones to deal with solutions with conflicts have been added.

Infeasibility management

- The introduction of two kinds of infeasibilities, induced by the use of solutions with extra capacity (IC, Infeasibility by Capacities) and solutions with conflicts (IP, Infeasibility by Precedences), has been proposed. This feature has introduced diversification into the search process, making it more effective.
- The management of infeasibility on the problem has been studied in detail and different algorithms to eliminate both kinds of infeasibilities and produce feasible solutions out of non-feasible ones have been proposed.
- Several measures to evaluate the infeasibility of a given solution and guide and control the search process have been developed.
- A collection of examples to illustrate all new concepts related to infeasibility has been provided. The relationship between IC and IP infeasibilities have been analyzed and it has been concluded that IC infeasibility can always be transformed into IP infeasibility without changing the routing part of the solution but the reciprocal statement is not true. Both kinds of infeasibilities have been finally combined into a joint search process.

Development of heuristics

- Several heuristics based on different techniques (Variable Neighborhood Search, Simulated Annealing, Exterior Search, etc.) have been developed, emphasizing the

importance of the Hybridized Variable Neighborhood Search algorithm, the Exterior algorithm and the Hybridized Variable Neighborhood Search algorithm.

Computational experience

- An extensive computational experience has been presented. The performance of the proposed algorithms have been tested on different sized instances of the DTSPMS and the results obtained have been analyzed in detail.
- New instances with 132 orders have been generated to evaluate the performance of heuristics when applied to larger test examples. The parameter setting of heuristics has been performed on new instances generated for that purpose.

Comparison of results

- The results obtained running Variable Neighborhood Search algorithms with and without the use of solutions with extra capacity have been compared and it has been observed that the introduction of these intermediate non-feasible solutions always improves the algoritms performance. The Variable Neighborhood Search algorithm with the best performance has been the Hybridized Variable Neighborhood Search (HVNS), specifically designed for the DTSPMS. Using this algorithm the best results published in the literature by other authors have been significatively improved.
- HVNS has been observed to perform better than the Exterior Algorithm, but the best overall results have been provided by the algorithm that combines both techniques, the Exterior Hybridized Variable Neighborhood Search algorithm. Good quality solutions for all considered instances have been obtained using little running time and the best known solutions of several instances have been improved.

Main contributions

The main results of the thesis have been presented in the following national and international conferences:

- EURO XXII. Prague (Czech Republic). July 2007.
- XXX National Conference on Statistics and Operations Research. Valladolid (Spain). September 2007.
- ECCO XXI. Dubrovnik (Croatia). May 2008.

- IWOR 2008. Móstoles (Spain). June 2008.
- Transnova 2008. Santa Cruz de Tenerife (Spain). June 2008.
- FLINS 2008. Madrid (Spain). September 2008.
- XXXI National Conference on Statistics and Operations Research. Murcia (Spain). February 2009.
- ECCO XXII. Jerusalem (Israel). May 2009.

The following publications have been originated from the thesis:

- A. Felipe, M.T. Ortuño and G. Tirado (2008) Neighborhood structures to solve the double traveling salesman problem (TSP) with multiple stacks using local search. *World Scientific Proceedings Series on Computer Engineering and Information Science I*, 701-706.
- A. Felipe, M.T. Ortuño and G. Tirado (2009) New neighborhood structures for the Double Traveling Salesman Problem With Multiple Stacks. *TOP* 17, 190-213.
- A. Felipe, M.T. Ortuño and G. Tirado (2009) The Double Traveling Salesman Problem With Multiple Stacks: A Variable Neighborhood Search Approach. *Computers and Operations Research* 36, 2983-2993.

Bibliografía

Aarts, E.H.L. y J. Lenstra (1997) Local Search in Combinatorial Optimisation. *Wiley*, Chichester.

Alegre Martínez, J.F. (2002) Diseño de metaheurísticos modernos para la resolución de problemas combinatorios. Aplicaciones a modelos logísticos de la industria del automóvil. *Tesis Doctoral*, Depto. de Estadística e Investigación Operativa, Fac. CC. Matemáticas, UCM, Madrid.

Alonso, A., P. Detti, L.F. Escudero y M.T. Ortuño (2003) On Dual Based Lower Bounds for the Sequential Ordering Problem with Precedences and Due Dates. *Annals of Operations Research* 124, 111-131.

Alshamrani, A., K. Mathur y R.H. Ballou (2007) Reverse logistics: Simultaneous design of delivery routes and return strategies. *Computers and Operations Research* 34, 595-619.

Andreatta, A. y C. Ribeiro (2002) Heuristics for the phylogeny problem. *Journal of Heuristics* 8, 429-447.

Anily, S. y G. Mosheiov (1994) The traveling salesman problem with delivery and backhauls. *Operations Research Letters* 16, 11-18.

Applegate, D.L., R.E. Bixby, V. Chvátal y W.J. Cook (2006) The traveling salesman problem: a computational study. *Princeton University Press.*

Archetti, C., A. Hertz y M.G. Speranza (2005) A tabu search algorithm for the split delivery vehicle routing problem. *Transportation Science* 39, 182-187.

Assad, A.A. y B.L. Golden (1995) Arc routing methods and applications. En M.O. Ball et al., eds., *Handbooks in OR and MS Volume 8*, Elsevier, Amsterdam, 375-483.

Audet, C., J. Brimberg, P. Hansen y N. Mladenović (2000) Pooling problem: alternate formulation and solution methods. *Les Cahiers du GERAD*, G-2000-23.

Baldacci, R., E.A. Hadjiconstantinou y A. Mingozzi (2003) An exact algorithm for

the traveling salesman problem with deliveries and collections. *Networks* 42, 26-41.

Barbarosoglu, G. y D. Ozgur (1999) A tabu search algorithm for the vehicle routing problem. *Computers and Operations Research* 26, 255-270.

Barnhart, C., E.L. Johnson, G.L. Nemhauser, M.W.P. Savelsbergh y P.H. Vance (1998) Branch-and-price: column generation for huge integer programs. *Operations Research* 46, 316-329.

Beasley, D., D.R. Bull y R.R. Martin (1993) An overview of genetic algorithms: Part 1, Fundamentals. *University Computing* 15, 58-69.

Belacel, N., P. Hansen y N. Mladenovic (2002) Fuzzy J-means: a new heuristic for fuzzy clustering. *Pattern Recognition* 35(10), 2193-2200.

Belenguer, J.M. y E. Benavent (2003) A cutting plane algorithm for the Capacitated Arc Routing Problem. *Computers and Operations Research* 30(5), 705-728.

Beltrami, E.J. y L.D. Boldin (1974) Networks and vehicle routing for municipal waste collection. *Networks* 4, 9-32.

Benavent, E., V. Campos, A. Corberán y E. Mota (1992) The Capacitated Arc Routing Problem. Lower Bounds. *Networks* 22, 669-690.

Bent, R. y P. Van Hentenryck (2003) A Two-Stage Hybrid Algorithm for Pickup and Delivery Vehicle Routing Problems with Time Windows. *Proceedings of the International Conference on Constraint Programming (CP-2003)*, Lecture Notes in Computer Science 2833, 123-137.

Bent, R. y P. Van Hentenryck (2004) A Two-Stage Hybrid Local Search for the Vehicle Routing Problem with Time Windows. *Transportation Science* 38(4), 515-530.

Bianchessi, N. y G. Righini (2007) Heuristic algorithms for the vehicle routing problem with simultaneous pick-up and delivery. *Computers and Operations Research* 34(2), 578-594.

Blum, C. y M. Sampels (2004) An ant colony optimization algorithm for shop scheduling problems. *J. Math. Model. Algorithms* 3(3), 285-308.

Bräysy, O. (2003) A Reactive Variable Neighborhood Search for the Vehicle-Routing Problem with Time Windows. *INFORMS Journal of Computing* 15(4), 347-368.

Brimberg, J., P. Hansen, K.W. Lih, N. Mladenović y M. Breton (2003) An oil pipeline design problem. *Operations Research* 51(2), 228-239.

Brimberg, J., P. Hansen, N. Mladenović y É. Taillard (2000) Improvements and comparison of heuristics for solving the multisource Weber problem. *Operations Research*

48(3), 444-460.

Brimberg, J. y N. Mladenović (1996) A variable neighborhood algorithm for solving the continuous location-allocation problem. *Studies in Locational Analysis* 10, 1-12.

Burke, E.K., P. De Causmaecker, S. Petrović y G.V. Bergue (2001) Variable neighbourhood search for nurse rostering. *MIC'2001*, Porto, 755-760.

Burke, E.K., P. Cowling y R. Keuthen (2001) Effective local and guided variable neighbourhood search methods for the asymmetric traveling salesman problem. *Lecture Notes in Computer Science* 2037, 203-212.

Canuto, S., M. Resende y C. Ribeiro (2001) Local search with perturbations for the prize-collection Steiner tree problem in graphs. *Networks* 38, 50-58.

Caporossi, G., D. Cvetković, I. Gutman y P. Hansen (1999) Variable neighborhood search for extremal graphs 2: Finding graphs with extremal energy. *Journal of Chemical Information and Computer Science* 39, 984-996.

Caporossi, G., I. Gutman y P. Hansen (1999) Variable neighborhood search for extremal graphs 4: Chemical trees with extremal conectivity index. *Computers and Chemistry* 23, 469-477.

Caporossi, G. y P. Hansen (2000) Variable Neighbourhood Search for extremal graphs 1: The AutoGraphiX system. *Discrete Mathematics* 212, 29-44.

Carrabs, F., R. Cerulli y J.-F. Cordeau (2007a) An additive branch-and-bound algorithm for the pickup and delivery traveling salesman problem with LIFO or FIFO loading. *INFOR* 45, 223-238.

Carrabs, F., J.-F. Cordeau y G. Laporte (2007b) Variable neighbourhood search for the pickup and delivery traveling salesman problem with LIFO loading. *INFORMS Journal on Computing* 19, 618-632.

Carraway, R.L., T.L. Morin y H. Moskowitz (1990) Generalized dynamic programming for stochastic combinatorial optimisation. *Operations Research* 37, 819-829.

Casazza, M., A. Ceselli y M. Nunkesser (2009) Efficient algorithms for the DTSPMS. *Proceedings of Cologne Twente Workshop in Combinatorial Optimization 2009.*

Cassani, L (2004) Algoritmi euristici per il TSP with rear-loading. *Degree thesis, DTI-Università di Milano.*

Cassani, L. y G. Righini (2004) Heuristic algorithms for the TSP with rear-loading. En *Proceedings of the 35th Annual Conference of the Italian Operations Research Society (AIRO XXXV)*, Lecce, Italy.

Ĉerný, V. (1985) Thermodynamical Approach to the Traveling Salesman Problem: An Efficient Simulation Algorithm. *Journal of Optimization Theory* 45, 41-51.

Ceselli, A., G. Righini y G. Tirado (2009) Mathematical programming algorithms for the double TSP with multiple stacks. En *Proceedings of The 22nd Conference of the European Chapter on Combinatorial Optimization (ECCO XXII)*, Jerusalem, Israel.

Charon, I. y O. Hudry (2007) A survey on the linear ordering problem for weighted or unweighted tournaments. *4OR* 5, 5-60.

Christofides, N. (1976) Worst-case analysis of a new heuristic for the travelling salesman problem. *Research report* 388, G.S.I.A., Carnegie Mellon University, Pittsburgh, PA.

Christofides, N. (1985) Vehicle routing. En E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan y D.B. Shmoys, eds., *The Traveling Salesman Problem*, Wiley, Chichester, UK.

Christofides, N., A. Mingozzi y P. Toth (1979) The Vehicle Routing Problem. En N. Christofides, A. Mingozzi, P. Toth y C. Sandi, eds., *Combinatorial Optimization*, Wiley.

Clarke, G. y J.W. Wright (1964) Scheduling of Vehicles from a Central Depot to a Number of Delivery Points. *Operations Research* 12, 568-581.

Cook, W. y J.L. Rich (1999) A parallel cutting plane algorithm for the vehicle routing problem with time windows. Technical report, *Computational Applied Mathematics* Rice University, Houston, TX.

Cordeau, J.-F., M. Iori, G. Laporte y J. Salazar-González (2009) A Branch-and-Cut Algorithm for the Pickup and Delivery Traveling Salesman Problem with LIFO Loading. To appear.

Cordeau, J.-F., G. Laporte y A. Mercier (2000) A unified tabu search heuristic for vehicle routing problems with time windows. Technical report CRT-00-03, *Centre for Research on Transportation*, Montreal, Canada.

Cordeau, J.-F., G. Laporte, J.-Y. Potvin y M.W.P. Savelsbergh (2006) Transportation on demand. En C. Barnhart y G. Laporte, eds., *Transportation*, Elsevier, Amsterdam.

Cordone, R. y R.W. Calvo (2001) A heuristic for the vehicle routing problem with time windows. *Journal of Heuristics* 7(2), 107-129.

Costa, D. y A. Hertz (1997) Ants can color graphs. *Journal of the Operational Research Society* 48, 295,305.

Costa, M.C., F.R. Monclar y M. Zrikem (2001) Variable neighborhood search for the optimization of cable layout problem. *Journal of Intelligent Manufacturing (Kluwer)*

13(5).

Côté, J.F., M. Gendreau y J.Y. Potvin (2009) An Effective Heuristic for the Pickup and Delivery Traveling Salesman Problem with LIFO Loading and Multiple Stacks. *Proceedings of Les Journées de l'optimisation 2009*, Montreal.

Crainic, T., M. Gendreau, P. Hansen, N. Hoeb y N. Mladenović (2001) Parallel variable neighborhod search for the p-median. En *Proceedings of the 4th Metaheuristics International Conference*, J.P. de Sousa Ed. Porto, MIC 2001, 595-599.

Crispim, J. y J. Brandao (2001) Reactive Tabu Search and Variable Neighbourhood Descent applied to the vehicle routing problem with backhauls. En *Proceedings of the 4th Metaheuristics International Conference*, J.P. de Sousa Ed. Porto, MIC 2001, 631-636.

Croes, G.A. (1958) A method for solving traveling salesman problems. *Operations Research* 6, 791-812.

Dantzig, G., R. Fulkerson y S. Johnson (1954) Solution of a large-scale traveling-salesman problem. *Operations Research* 2, 393-410.

Dantzig, G.B. y J.H. Ramser (1959) The Truck Dispatching Problem. *Management Science* 6(1), 81-91.

Dantzig, G.B. y P. Wolfe (1960) Decomposition Principle for Linear Programs. *Operations Research* 8(1), 101-111.

Dantzig, G.B. y P. Wolfe (1961) The Decomposition Algorithm for Linear Programs. *Econometrica* 29(4), 767-778.

Davidović, T., P. Hansen y N. Mladenović (2001) Variable neighborhood search for multiprocessor scheduling with communications delays. *MIC'2001*, Porto, 737-741.

Desaulniers, G., J. Desrosiers y M.M. Solomon (Eds.) (2005) Column Generation. *Springer*, Berlín.

Desrosiers, J. y F. Soumis (1989) A Column Generation Approach to the Urban Transit Crew Scheduling Problem. *Transportation Science* 23, 1-13.

Desrosiers, J., F. Soumis y M. Desrochers (1984) Routing with Time Windows by Column Generation. *Networks* 14, 545-565.

Díaz, A. (1996) Optimización heurística y Redes Neuronales en Dirección de Operaciones e Ingeniería. *Paraninfo*, Madrid.

Doerner, K., G. Fuellerer, M. Gronalt, R. Hartl y M. Iori (2007) Metaheuristics for vehicle routing problems with loading constraints. *Networks* 49, 294-307.

Dorigo, M. (1992) Optimization, learning and natural algorithms. *Tesis doctoral (en italiano)*, Dipartimento di Elettronica, Politecnico di Milano, Italia.

Dorigo, M., V. Maniezzo y A. Colorni (1991) Positive feedback as a search strategy. *Tech. Report 91-016*, Dipartimento di Elettronica, Politecnico di Milano, Italia.

Dorigo, M., V. Maniezzo y A. Colorni (1996) Ant system: Optimization by a colony of cooperating agents. *IEEE Trans Sys, Man Cybernetics* 26, 29-41.

Dorigo, M. y T. Stützle (2004) Ant Colony Optimization. *MIT Press*, Cambridge, MA.

Dowsland, K.A. (1995) Simulated Annealing. En *Modern Heuristic Techniques for Combinatorial Problems*, C.R. Reeves (ed), McGraw-Hill, UK.

Dowsland, K.A. y B.A. Díaz (2001) Diseño de heurísticas y fundamentos del Recocido Simulado. *Inteligencia Artificial, Revista Iberoamericana de Inteligencia Artificial* 20, 34-52.

Dror, M., G. Laporte y P. Trudeau (1989) Vehicle routing with sotochastic demands: Properties and solution frameworks. *Transportation Science* 23, 166-176.

Dror, M., G. Laporte y P. Trudeau (1994) Vehicle routing with split deliveries. *Discrete Applied Mathematics* 50, 239-254.

Dror, M. y P. Trudeau (1989) Savings by split delivery routing. *Transportation Science* 23, 141-145.

Dror, M. y P. Trudeau (1990) Split delivery routing. *Naval Research Logistics* 37, 383-402.

Dumitrescu, I., S. Ropke, J.-F. Cordeau y G. Laporte (2008) The traveling salesman problem with pickups and deliveries: polyhedral results and branch-and-cut algorithm. *Mathematical Programming, Series A*, Forthcoming. ISSN 1436-4646, doi:10.1007/s10107-008-0234-9.

Escudero, L.F. y M.T. Ortuño (1997) On due-date based valid cuts for the sequential ordering problem. *TOP* 5(1), 159-166.

Feo, T. y M. Resende (1989) A Probabilistic Heuristic for a Computationally Difficult Set Covering Problem. *Operations Research Letters* 8, 67-71.

Feo, T. y M. Resende (1995) Greedy randomized adaptive search procedures. *Journal of Global Optimization* 6, 109-133.

Festa, P., P. Pardalos, M. Resende y C. Ribeiro (2001) GRASP and VNS for max-cut. En *Proceedings of the 4th Metaheuristics International Conference*, J.P. de Sousa Ed. Porto, MIC 2001, 371-376.

Fischetti, M. y P. Toth (1989) An additive bounding procedure for combinatorial optimization problems. *Operations Research* 37, 319-328.

Fisher, M.L. (1981) The lagrangian relaxation method for solving integer programming problems. *Management Science* 27, 1-18.

Fisher, M.L. (1985) An applications oriented guide to lagrangian relaxation. *Interfaces* 15, 10-21.

Fisher, M.L. (2004a) The Lagrangian Relaxation Method for Solving Integer Programming Problems. *Management Science* 50(12), 1861-1871.

Fisher, M.L. (2004b) Comments of "The Lagrangian Relaxation Method for Solving Integer Programming Problems". *Management Science* 50(12), 1872-1874.

Fisher, M.L., W.D. Northup y J.F. Shapiro (1975) Using duality to solve discrete optimization problems: Theory and computational experience. *Mathematical Programming Study* 3, 56-94.

Fleszar, K. y K.S. Hindi (2002) New heuristics for one-dimensional bin-packing. *Computers and Operations Research* 29(7), 821-839.

Gagné, C., W.L. Price y M. Gravel (2002) Comparing an ACO algorithm with other heuristics for the single machine scheduling problem with sequence-dependent setup times. *J. Oper. Res. Soc.* 53, 895-906.

Gambardella, L.M., E.D. Taillard y G. Agazzi (1999) MACS-VRPTW: a multiple ant colony system for vehicle routing problems with time windows. En D. Corne, M. Dorigo y F. Glover (eds.), *New Ideas in Optimization*, McGraw-Hill, London, UK, 63-76.

García López, F., M.B. Melián Batista, J.A. Moreno Pérez y J.M. Moreno Vega (2001) The parallel variable neighborhood search for the p-median problem. *Journal of Heuristics* 8, 377-390.

Garey, M.R. y D.S. Johnson (1979). Computers and Intractability: A Guide to the Theory of NP-Completeness *Freeman*, San Francisco.

Gendreau, M. (2002) Recent Advances in Tabu Search. *Essays and Surveys in Metaheuristics*, C.C. Ribeiro and P. Hansen (eds.), Kluwer Academic Publishers, 369-377.

Gendreau, M., P. Dejax, D. Feillet y C. Gueguen (2003) Vehicle routing with split deliveries and time windows. *Odysseus 2003*, Palermo, Italia.

Gendreau, M., A. Hertz y G. Laporte (1994) A tabu search heuristic for the vehicle routing problem. *Management Science* 40, 1276-1290.

Gendreau, M., M. Iori, G. Laporte y S. Martello (2006) A heuristic algorithm for

a routing and container loading problem. *Transportation Science* 40, 342-350.

Gendreau, M., M. Iori, G. Laporte y S. Martello (2008) A tabu search heuristic for the vehicle routing problem with two-dimensional loading constraints. *Networks*, 51(1), 4-18.

Gendreau, M., G. Laporte and J.Y. Potvin (2002) Metaheuristics for the capacitated VRP, in: P. Toth and D. Vigo (eds.) *The Vehicle Routing Problem, SIAM Monographs on Discrete Mathematics and Applications*, Philadelphia, 129-154.

Gendreau, M., G. Laporte y R. Séguin (1995) An exact algorithm for the vehicle routing problem with stochastic demands and costumers. *Transportation Science* 29, 143-155.

Gendreau, M., G. Laporte y R. Séguin (1996) Stochastic Vehicle Routing. *European Journal of Operational Research* 88, 3-12.

Gendreau, M., G. Laporte y D. Vigo (1999) Heuristics for the traveling salesman problem with pickup and delivery. *Computers and Operations Research* 26, 699-714.

Geoffrion, A.M. (1974) Lagrangean relaxation for integer programming. *Mathematical Programming Study* 2, 82-114.

Ghiani, G., A. Hertz y G. Laporte (2000) Recent algorithmic advances for arc routing problems. *Les Cahiers du GERAD*, G-2000-40.

Gilmore, P.C. y R.E. Gomory (1961) A Linear Programming Approach to the Cutting Stock Problem. *Operations Research* 9, 849-859.

Gilmore, P.C. y R.E. Gomory (1963) A Linear Programming Approach to the Cutting Stock Problem: Part II. *Operations Research* 11, 863-888.

Glover, F. (1977) Heuristics for integer programming using surrogate constraints. *Decision Sciences* 8, 156-166.

Glover, F. (1989) Tabu search. Part I. *ORSA Journal of Computing* 1, 190-206.

Glover, F. (1990) Tabu search. Part II. *ORSA Journal of Computing* 1, 4-32.

Glover, F. (1998) A Template for Scatter Search and Path Relinking. En J.-K. Hao, E. Lutton, E. Ronald, M. Schoenauer y D. Snyers (Eds.) *Artificial Evolution, Lecture Notes in Computer Science* 1363, Springer-Verlag, 13-54.

Glover, F. y G.A. Kochenberger (2003) . *Handbook of Metaheuristics*, Kluwer, Boston.

Glover, F. y M. Laguna (1993) Tabu Search. En *Modern Heuristic Techniques for Combinatorial Problems*. C.R Reeves (ed.), Blackwell, 70-150.

Glover, F. y M. Laguna (1997) Tabu Search. *Kluwer Academic Publishers*, Norwell, MA.

Glover, F. y M. Laguna (2002) Tabu Search. En *Handbook of Applied Optimization*, P.M. Pardalos y M.G.S. Resende (eds), Oxford University Press, 194-208.

Glover, F., M. Laguna y R. Martí (2000) Fundamentals of Scatter Search and Path Relinking. *Control and Cybernetics* 29(3), 653-684.

Glover, F., M. Laguna, É. Taillard y D. de Werra (eds.) (1993) Tabu Search. *Annals of Operations Research* 41, J.C. Baltzer Science Publishers, Basel, Switzerland.

Glover, F., É. Taillard y D. de Werra (1993) A User's Guide to Tabu Search. *Annals of Operations Research* 41, 3-28.

Goldberg, D.E. (1989) Genetic Algorithms in Search, Optimization, and Machine Learning. Reading, MA, *Addison-Wesley.*

Goldberg, D.E., K. Zakrzewski, B. Sutton, R. Gadient, C. Chang, P. Gallego, B. Miller y E. Cantú-Paz (1997) Genetic Algorithms: A Bibliography. IlliGAL Report no. 97011, *Illinois Genetic Algorithms Laboratory*, University of Illinois at Urbana-Champaign.

Golden, B.L. y A.A. Assad (1988) Vehicle Routing: Methods and Studies. *North-Holland*, Amsterdam.

Golden, B.L. y R.T. Wong (1981) Capacitated Arc Routing Problems. *Networks* 1, 305-315.

Gueguen, C. (1999) Méthodes de résolution exacte pour les problèmes de tournées de véhicules. *PhD thesis, Ecole Centrale Paris.*

Hansen, P. y N. Mladenović (1997) Variable neighborhood search for the p-median. *Location Science 5*, 207-226.

Hansen, P. y N. Mladenović (1999) An introduction to variable neighbourhood search. En *Metaheuristics: Advances and Trends in Local Search Paradigms for Optimization.* S. Voss, S. Martello, I. H. Osman y C. Roucairol, Eds. Kluwer Academic Publisers, 433-458.

Hansen, P. y N. Mladenović (2000a) Recherche a Voisinage Variable. *Les Cahiers de GERAD G2000-08.* Montréal. Canadá.

Hansen, P. y N. Mladenović (2000b) A separable approximation dynamic programming algorithm for economic dispatch with transmission losses. *GERAD report.* G-2000-31.

Hansen, P. y N. Mladenović (2001a) Variable Neighbourhood Search: Principles and Applications. *European Journal of Operational Research* 130, 449-467.

Hansen, P. y N. Mladenović (2001b) Developments of variable neighbourhood search. En *Essays and Surveys in Metaheuristics*. C. Ribero y P. Hansen, Eds. Kluwer, 415-440.

Hansen, P. y N. Mladenović (2001c) Industrial applications of the variable neighbourhood search metaheuristics. En *Decisions and Control in Management Science*. G. Zaccour, Ed. Kluwer, 261-274.

Hansen, P. y N. Mladenović (2001d) J-means: A new local search heuristic for minimum sum-of-squares clustering. *Pattern Recognition* 34, 405-413.

Hansen, P. y N. Mladenović (2002) Variable Neighbourhood Search. En *Handbook of Applied Optimization*. Panos M. Pardalos y Mauricio G. C. Resende Kochenberger, Eds.

Hansen, P. y N. Mladenović (2003) Variable Neighbourhood Search. En *Handbook of Metaheuristics*. Fred Glover y Gary A. Kochenberger, Eds. Kluwer. Capítulo 6.

Hansen, P., N. Mladenović y J.A. Moreno Pérez (2003) Búsqueda de Entorno Variable. *Revista Iberoamericana de Inteligencia Artificial* 19, 77-92.

Hansen, P., N. Mladenović y D. Pérez Brito (2001) Variable neighborhood decomposition search. *Journal of Heuristics* 7(4), 335-350.

Hansen, P., N. Mladenović y D. Urošević (2001) Variable neighborhood search for the maximum clique. *Les Cahiers du GERAD*, G-2001-08.

Hart, J.P. y A.W. Shogan (1987) Semi-Greedy Heuristics: An Empirical Study. *Operations Research Letters* 6, 107-114.

Healy, P. y R. Moll (1995) A new extension of local search applied to the dial-a-ride problem. *European Journal of Operational Research* 83, 83-104.

Held, M. y R.M. Karp (1970) The traveling salesman problem and minimum spanning trees. *Operations Research* 18, 1138-1162.

Held, M. y R.M. Karp (1971) The traveling salesman problem and minimum spanning trees: Part II. *Mathematical Programming* 1, 6-25.

Held, M., P. Wolfe y H.D. Crowder (1974) Validation of subgradient optimization. *Mathematical Programming* 6, 62-88.

Hernández-Pérez, H. y J.J. Salazar-González (2004) Heuristics for the one-commodity pick-up and delivery Traveling Salesman Problem. *Transportation Science* 38, 245-255.

Hernández-Pérez, H. y J.J. Salazar-González (2007) The one-commodity pickup-and-delivery traveling salesman problem: Inequalities and algorithms. *Networks* 50(4), 258-272.

Hertz, A. y D. de Werra (1991) The Tabu Search Metaheuristic: How We Used It. *Annals of Mathematics and Artificial Intelligence* 1, 111-121.

Holland, J.H. (1975) Adaptation in Natural and Artificial Systems. Ann Arbor, MI: University of Michigan Press. Hay disponible una segunda edición: Cambridge, MA: The MIT Press 1992.

Homberger, J. y H. Gehring (1999) Two evolutionary metaheuristics for the vehicle routing problem with time windows. *INFOR* 37, 297-318.

Huang, X. y G. Wang (2007) A modified golumbic algorithm for permutation graphs in VLSI. *ASICON '07*, 237-240.

Iori, M., J. Salazar-González y D. Vigo (2007) An exact approach for the vehicle routing problem with two-dimensional loading constraints. *Transportation Science* 41(2), 253-264.

Jünger, M., G. Reinelt y G. Rinaldi (1995) The Traveling Salesman Problem. En M.O. Ball, T.L. Magnanti, C.L. Monma y G.L. Nemhauser (eds),*Handbook in Operations Research and Management Science*, volumen 7, Network Models, North-Holland, Amsterdam, 225-330.

Kalantari, B., A.V. Hill y S.R. Arora (1985) An algorithm for the traveling salesman problem with pickup and delivery customers. *European Journal of Operational Research* 22, 377-386.

Kirkpatric, S., C.D. Gellat y M.P. Vecchi (1983) Optimization by simulated annealing. *Science* 220, 671-680.

Knight, K. y J. Hofer (1968) Vehicle scheduling with timed and connected calls: A case study. *Operational Research Quartely* 19, 299-310.

Koulamas, C., S.R. Anthony y R. Jane (1994) A survey of simulated annealing applications to operations reserach problems. *Omega* 22, 41-56.

Kovačević, V., M. Čangalović, M. Ašić, D. Dražić y L. Ivanović (1999) Tabu search methodology in global optimization. *Computers and Mathematics with Applications* 37, 125-133.

Kubale, M. (2004) Graph colorings. *American Mathematical Society*, Providence, Rhode Island.

Kubo, M. y H. Kasugai (1990) Heuristic algorithms for the single vehicle dial-a-ride problem. *Journal of the Operations Research Society of Japan* 33, 354-365.

Kulkarni, R.V. y P.R. Bhave (1985) Integer Programming Formulations of Vehicle

Routing Problems. *European Journal of Operational Research* 20, 58-67.

Lacomme, C. Prins y W. Ramdane-Chérif (2001) A genetic algorithm for the Capacitated Arc Routing Problem and its extensions. En E.J.W. Boers et al., eds., *Applications of evolutionary computing*, Lecture Notes in Computer Science 2037, Springer, Berlin, 473-483.

Ladany, S.P. y A. Mehrez (1984) Optimal routing of a single vehicle with loading constraints. *Transportation Planning and Technology* 8, 301-306.

Laguna, M. y R. Martí (2003) Scatter Search: Methodology and implementations in C. *Kluwer*, Academic Publishers, Boston.

Lambert, V., G. Laporte y F. Louveaux (1993) Designing collection routes through bank branches. *Computers and Operations Research* 20, 783-791.

Laporte, G., F.V. Louveaux y H. Mercure (1989) Models and exact solutions for a class of stochastic location-routing problems. *European Journal of Operational Resarch* 39, 71-78.

Laporte, G. e I.H. Osman (1996) Metaheuristics in Combinatorial Optimization. *Annals of Operations Research* 63, J.C. Baltzer Science Publishers, Basel, Switzerland.

Lawler, E.L., J.K. Lenstra, A.H.G. Rinnooy Kan y D.B. Shmoys (1985) The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization. *Wiley*, Chichester, UK.

Li, Y., P.M. Pardalos y M.G.C. Resende (1994) A Greedy Randomized Adaptative Search Procedure of the Quadratic Assignment Problem. En P.M. Pardalos y H. Wolkowicz (eds.), *Quadratic Assignment and Related Problems*, DIMACS vol. 16, American Mathematical Society.

Madsen, O.B.G. (1976) Optimal scheduling of trucks - A routing problem with tight due times for delivery. En H. Strobel, R. Genser y M. Etschmaier, eds., *Optimization applied to transportation systems*, International Institute for Applied System Analysis, Laxenburgh, Austria, 126-136.

Madsen, O.B.G., N. Kohl, J. Desrosiers, M. Solomon y F. Soumis (1999) 2-Path Cuts for the Vehicle Routing Problem with Time Windows. *Transportation Science* 13, 101-116.

Magnanti, T.L., J.F. Shapiro y M.H. Wagner (1976) Generalized linear programming solves the dual. *Management Science* 22, 1195-1203.

Maniezzo, V. y A. Colorni (1999) The ant system applied to the quadratic assignment problem. *IEEE Trans. Data Knowledge Eng.*, 11(5), 769-778.

Martins, S.L., M.G.C. Resende, C.C. Ribeiro y P. Pardalos (2000) A parallel GRASP for the Steiner tree problem in graphs using a hybrid local search strategy. *Journal of Global Optimization* 17, 267-283.

Merkle, D., M. Middendorf y H. Schmeck (2002) Ant colony optimization for resource-constrained project scheduling. *IEEE Trans. Evol. Comput.* 6(4), 333-346.

Mitchell, M. (1996) An Introduction to Genetic Algorithms. *MIT Press.*

Mladenović, N. (1995) A Variable Neighbourhood Algorithm. A New Metaheuristic for Combinatorial Optimization. Abstracts of papers presented at *Optimization Days*. Montréal.

Mladenović, N. y P. Hansen (1997) Variable Neighbourhood Search. *Computers and Operations Research* 24, 1097-1100.

Mladenović, N., M. Labbe y P. Hansen (2003) Solving the p-center problem with tabu search and variable neighborhood search. *Networks* 42(1), 48-64.

Mladenović, N., J. Petrović, V. Kovaĉevic-Vujčić y M. Čangalović (2003) Solving spread spectrum radar polyphase code design problem by tabu search and variable neighborhood search. *European Journal of Operational Research* 151, 389-399.

Mladenović, N. y D. Urošević (2001) Variable neighborhood search for the k-cardinality. *Computers and Operations Research* 31(8), 1205-1213.

Mosheiov, G. (1994) The traveling salesman problem with pick-up and delivery. *European Journal of Operational Research* 79, 299-310.

Mullaseril, P.A., M. Dror y J. Leung (1997) Split-delivery routing in livestock feed distribution. *Journal of the Operational Research Society* 48, 107-116.

Ochi, L.S., M.B. Silva y L. Drummond (2001) Metaheuristics based on GRASP and VNS for solving traveling purchaser problem. En *Proceedings of the 4th Metaheuristics International Conference*, J.P. de Sousa Ed. Porto, MIC 2001, 489-494.

Osman, I.H. (1993) Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. *Annals of Operations Research* 41, 421-451.

Pacheco Bonrostro, J. (1997a) Heurístico para los problemas de rutas con carga y descarga en sistemas LIFO. *SORT, Statistics and Operations Research Transactions* 21, 153-175.

Pacheco Bonrostro, J. (1997b) Metaheuristic based on a simulated annealing process for one vehicle pick-up and delivery problem in LIFO unloading systems. En *Proceedings of the Tenth Meeting of the European Chapter of Combinatorial Optimization (ECCO X)*,

Tenerife, Spain.

Petersen, H. (2006) Heuristic Solution Approaches to the Double TSP with Multiple Stacks. *Technical Report*, Centre for Traffic and Transport, Technical University of Denmark.

Petersen, H., C. Archetti y M.G. Speranza (2008) Exact Solutions to the Double Travelling Salesman Problem with Multiple Stacks. *Technical report.*

Petersen, H. y O.B.G. Madsen (2009) The Double Travelling Salesman Problem with Multiple Stacks - Formulation and Heuristic Solution Approaches. *European Journal of Operational Research* 198(1), 139-147.

Psaraftis, H. N. (1983a) Analysis of an O(n^2) heuristic for the single vehicle many-to-many Euclidean dial-a-ride problem. *Transportation Research B.* 17, 133-145.

Psaraftis, H. N. (1983b) An exact algorithm for the single-vehicle, many-to-many dial-a-ride problem with time windows. *Transportation Science* 17, 351-357.

Pullen, H. y M. Webb (1967) A computer application to a transport scheduling problem. *Computer Journal* 10, 10-13.

Reeves, C.R. (1995) Modern Heuristic Techniques for Combinatorial Problems. McGraw-Hill, UK.

Rego, C. (1998) A subpath ejection method for the vehicle routing problem. *Management Science* 44, 1447-1459.

Rego, C. y C. Roucairol (1996) A parallel tabu search algorithm using ejection chains for the vehicle routing problem. *Meta-Heuristics: Theory and Applications*, I.H. Osman y J.P. Kelly (eds), Kluwer, Boston, MA, 661-675.

Reimann, M., K. Doerner y R.F. Hartl (2004) D-ants: savings based ants divide and conquer the vehicle routing problems. *Computers and Operations Research* 31(4), 563-591.

Renaud, J., F.F. Boctor y G. Laporte (2002) Perturbation heuristics for the pickup and delivery traveling salesman problem. *Computers and Operations Research* 29, 1129-1141.

Renaud, J., F.F. Boctor y J. Ouenniche (2000) A heuristic for the pickup and delivery traveling salesman problem. *Computers and Operations Research* 27, 905-916.

Ribeiro, C.C. y P. Hansen (eds.) (2002) Essays and Surveys in Metaheuristics. Kluwer Academic Publishers, Norwell, MA.

Ribeiro, C.C. y M.C. Souza (2002) Variable neighbourhood descent for the degree-constrained minimum spanning tree problem. *Discrete Applied Mathematics* 118, 43-54.

Ribeiro, C.C., E. Uchoa y R. Werneck (2002) A hybrid GRASP with perturbations for the Steiner problem in graphs. *INFORMS Journal of Computing* 14, 228-246.

Roberts, D. (1998) Algorithms for stochastic vehicle routing problems. Ph.D. Thesis, Imperial College, University of London.

Roberts, D. y E. Hadjiconstantinou (1998) A computational approach to the vehicle routing problem with stochastic demands. P. Borne, M. Ksouri y A. El Kamel, *Computational Engineering in Systems Applications*, 139-144.

Rochat, Y. y E.D. Taillard (1995) Probabilistic diversification and intensification in local search or vehicle routing. *Journal of Heuristics* 1, 147-167.

Rodríguez, I., J.M. Moreno-Vega y J.A. Moreno Pérez (2003) Variable Neighborhood Tabu Search and its Application to the Median Cycle Problem. *European Journal of Operational Research* 151, 365-378.

Ruland, K.S. y E.Y. Rodin (1997) The pickup and delivery problem: faces and branch-and-cut algorithm. *Computers and Mathematics with Applications* 33, 1-13.

Savelsbergh, M. (1985) Local search for routing problems with time windows. *Annals of Operations Research* 4, 285-305.

Savelsbergh, M. (1990) An efficient implementation of local search algorithms for constrained routing problems. *European Journal of Operational Research* 47, 75-85.

Schiavinotto, T. y T. Stützle (2004) The Linear Ordering Problem: Instances, Search Space Analysis and Algorithms. *Journal of Mathematical Modelling and Algorithms* 3, 367-402.

Schrimpf, G., J. Schneider, H. Stamm-Wilbrandt y G. Dueck (2000) Record Breaking Optimization Results Using the Ruin and Recreate Principle. *Journal of Computational Physics* 159, 139-171.

Shapiro, J.F. (1979a) A survey of Lagrangian techniques for discrete optimization. *Ann. Discrete Math.* 5, 113-138.

Shapiro, J.F. (1979b) Mathematical Programming: Structures and Algorithms. *Wiley*, New York.

Shaw, P. (1997) A new local search algorithm providing high quality solutions to vehicle routing problems. *Technical Report,* Department of Computer Science, University of Strathclyde, Scotland.

Shaw, P. (1998) Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems. En M. Maher, J.-F. Puget, eds., *Principles and Practice of*

Constraint Programming - CP98, Lecture Notes in Computer Science, Springer-Verlag, New-York, 417-431.

Sierksma, G. y G.A. Tijssen (1998) Routing helicopters for crew exchanges on off-shore locations. *Annals of Operations Reserach* 76, 261-286.

Silva, M.B., L. Drummond y L.S. Ochi (2000) Variable Neighbourhood Search for the traveling purchaser problem. En *27th Int. Conference on Computational Industrial Engineering.*

Soriano, P. y M. Gendreau (1997) Fondements et applications des méthodes de recherche avec tabous. *RAIRO (Recherche opérationnelle)* 31, 133-159.

Stein, D. (1978) An asymptotic, probabilistic analysis of a routing problem. *Mathematics of Operations Research* 3, 89-101.

Stewart, W. y B. Golden (1983) Stochastic vehicle routing: A comprehensive approach. *European Journal of Operational Research* 14(3), 371-385.

Stützle, T. y H.H. Hoos (2000) MAX-MIN ant system. *Future Gen. Comput. Systems* 16(8), 889-914.

Suman, B. y P. Kumar (2006) A survey of simulated annealing as a tool for single and multiobjective optimization. *The Journal of the Operational Research Society* 57(10), 1143-1160.

Süral, H. y J.H. Bookbinder (2003) The single-vehicle routing problem with unrestricted backhauls. *Networks* 41, 127-136.

Taillard, E.D. (1993) Parallel iterative search methods for vehicle routing problems. *Newtorks* 23, 661-673.

Tillman, F. (1969) The multiple terminal delivery problem with probabilistic demands. *Transportation Science* 3, 192-204.

Tricoire, F., K.F. Doerner, R.F. Hartl y M. Iori (2008) Heuristic and Exact Algorithms for the Multi-Pile Vehicle Routing Problem. *Technical report DISMI*, Universitá di Modena e Reggio Emilia, enviado.

Toth, P. y D. Vigo (2003) The granular tabu search and its applications to the vehicle routing problem. *INFORMS Journal of Computing* 15, 333-346.

Van der Bruggen, L.J.J., J.K. Lenstra y P.C. Schuur (1993) Variable-depth search for the single-vehicle pickup and delivery problem with time windows. *Transportation Science* 27, 298-311.

Van Laarhoven, P.J.M. y E.H.L. Aarts (1992) Simulated Annealing: Theory and

Applications, Reidel Publishing Company.

Voss, S., S. Martello, I.H. Osman y C. Roucairol (eds.) (1999) Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization, Kluwer Academic Publishers, Norwell, MA.

Werra, de, D. y A. Hertz (1989) Tabu Search Techniques: A Tutorial and an Application to Neural Networks. *OR Spektrum* 11, 131-141.

Whitley, D. (1994) A Genetic Algorithm Tutorial. *Statistics and Computing* 4, 65-85.

Wilson, H. y N. Colvin (1977) Computer control of the Rochester dial-a-ride system. Technical Report R77-31, *Department of Civil Engineering*, MIT, Cambridge, MA.

Wilson, H., J. Sussman, H. Wang y B. Higonnet (1971) Scheduling algorithms for dial-a-ride systems. Technical Report USL TR-70-13, *Urban System Laboratory*, MIT, Cambridge, MA.

Wilson, H. y H. Weissberg (1977) Advanced dial-a-ride algorithms reserach project: Final report. Technical Report R76-20, *Department of Civil Engineering*, MIT, Cambridge, MA.

Xu, H., Z. Chen, S. Rajagopal y S. Arunapuram (2003) Solving a practical pickup and delivery problem. *Transportation Science* 37, 347-364.

Xu, J. y J.P. Kelly (1996) A network flow-based tabu search heuristic for the vehicle routing problem. *Transportation Science* 30, 379-393.

Principales páginas web

- Bibliografía sobre GA:
  ftp://ftp-illigal.ge.uiuc.edu/pub/papers/IlliGALs/97011.ps.Z
- Información sobre el TSP:
  http://www.tsp.gatech.edu/index.html
- Información sobre el VRP:
  http://neo.lcc.uma.es/radi-aeb/WebVRP/
- Web de Hanne L. Petersen:
  http://www.imm.dtu.dk/$\sim hlp$
- Web de Gregorio Tirado Domínguez:
  http://www.espaciotd.jazztel.es/dtspmsEn.htm

Glosario de siglas

- **1-PDMS:** 1-Pickup and Delivery problem with Multiple Stacks (Problema del viajante con Entrega y Recogida de mercancía y Múltiples Pilas).
- **1-SCR:** 1-Step Complete Reduction (Reducción Completa en 1 Etapa).
- **1-SR:** 1-Step Reduction (Reducción en 1 Etapa).
- **1-SRA:** 1-Step Reduction Assignments (Asignaciones para Reducción en 1 Etapa).
- **ACO:** Ant Colony Optimization (Algoritmos de Colonias de Hormigas).
- **ALNS:** Adaptative Large Neighborhood Search (Búsqueda en Entornos Grandes Adaptativa).
- **ARP:** Arc Routing Problem (Problema de Rutas por Arcos).
- **BPP:** Bin Packing Problem (Problema de Empaquetado).
- **BVNS:** Basic Variable Neighborhood Search (Búsqueda en Entorno Variable Básica).
- **CARP:** Capacitated Arc Routing Problem (Problema de Rutas por Arcos con Capacidades).
- **CCA:** Conflict Cuts Algorithm (Algoritmo de Cortes con Conflictos).
- **COL:** Graph Coloring Problem (Problema de Coloración de Grafos).
- **CS:** Complete Swap (Intercambio Entre Pilas).
- **CVRP:** Capacitated Vehicle Routing Problem (Problema de Rutas de Vehículos con Capacidades).
- **DSA:** Directed Stack Assignment (Asignación de Pilas Dirigida).
- **DTSPMS:** Double Traveling Salesman Problem with Multiple Stacks (Doble Problema del Viajante con Múltiples Pilas).

- **DTSPSS:** Double Traveling Salesman Problem with a Single Stack (Doble Problema del Viajante con una Sola Pila).
- **EHVNS:** Exterior Hybridized Variable Neighborhood Search (Búsqueda en Entorno Variable Híbrida Exterior).
- **EXT:** Exterior Algorithm (Algoritmo Exterior).
- **GA:** Genetic Algorithm (Algoritmo Genético).
- **GRAS:** Global Reduction with Adjustable Size (Reducción Global con Tamaño Ajustable).
- **GRASP:** Greedy Randomized Adaptative Search Procedure (Búsqueda Greedy Aleatorizada Adaptativa).
- **GRFS:** Global Reduction with Fixed Size (Reducción Global con Tamaño Fijo).
- **GVNS:** General Variable Neighborhood Search (Búsqueda en Entorno Variable General).
- **HVNS:** Hybridized Variable Neighborhood Search (Búsqueda en Entorno Variable Híbrida).
- **IC:** Infeasibility by Capacities (Infactibilidad por Capacidad).
- **IP:** Infeasibility by Precedences (Infactibilidad por Precedencias).
- **ISS:** In-Stack Swap (Intercambio Dentro de una Pila).
- **JMSI:** Generic Joint Multi Start and Intensification Algorithm (Algoritmo Genérico Conjunto de Multiarranque con Intensificación).
- **LIFO:** Last-In-First-Out (Último en Entrar, Primero en Salir).
- **LNS:** Large Neighborhood Search (Búsqueda en Entornos Grandes).
- **LOP:** Lineal Ordering Problem (Problema de Ordenación Lineal).
- **MCP:** Median Cycle Problem (Problema del Ciclo Mediana).
- **MDVRP:** Multi Depot Vehicle Routing Problem (Problema de Rutas de Vehículos con Múltiples Depósitos).
- **MP-VRP:** Multi-Pile Vehicle Routing Problem (Problema de Rutas de Vehículos con Múltiples Pilas).
- **MS:** Multi Start (Multi-Arranque).

- **MSI:** Generic Multi Start and Intensification algorithm (Algoritmo Genérico de Multiarranque con Intensificación).
- **MSR:** Multi-Step Reduction (Reducción Multi-Etapa).
- **MSRB:** Multi-Step Reduction with Backtracking (Reducción Multi-Etapa con Paso Atrás).
- **PR:** Path Relinking (Encadenamiento de Trayectorias).
- **PDP:** Pickup and Delivery Problem (Problema de Entrega y Recogida de mercancías).
- **PDPTW:** Pickup and Delivery Problem with Time Windows (Problema de Entrega y Recogida de mercancías con Ventanas de Tiempo).
- **PVNS:** Parallel Variable Neighborhood Search (Búsqueda en Entorno Variable Paralela).
- **PVRP:** Periodic Vehicle Routing Problem (Problema de Rutas de Vehículos Periódico).
- **R:** Reinserción (Reinsertion).
- ***r*-CSP:** r-Complete Stack Permutation (r-Permutación Completa en Pila).
- ***r*-RP:** r-Route Permutation (r-Permutación en Ruta).
- ***r*-SP:** r-Stack Permutation (r-Permutación en Pila).
- **RR:** Route Reinsertion (Reinserción en Ruta).
- **RS:** Route Swap (Intercambio en Ruta).
- **RVNS:** Reduced Variable Neighborhood Search (Búsqueda en Entorno Variable Reducida).
- **SA:** Simulated Annealing (Temple Simulado).
- **SAR:** Simulated Annealing with Reinsertion local search (Temple Simulado con búsqueda local mediante Reinserción).
- **SDVRP:** Split Delivery Vehicle Routing Problem (Problema de Rutas de Vehículos con carga Dividida).
- **SEC:** Subcycle Elimination Constraints (Restricciones de Eliminación de Subciclos).
- **SPLP:** Simple Plant Location Problem (Problema Simple de Localización de Plantas).

- **SR:** Stack Reinsertion (Reinserción en Pila).
- **SS:** Scatter Search (Búsqueda Dispersa).
- **SVNS:** Skewed Neighborhood Search (Búsqueda en Entorno Variable Sesgada).
- **SVRP:** Stochastic Vehicle Routing Problem (Problema de Rutas de Vehículos Estocástico).
- **TS:** Tabu Search (Búsqueda Tabú).
- **TSP:** Traveling Salesman Problem (Problema del Viajante).
- **TSPB:** Traveling Salesman Problem with Backhauls (Problema del Viajante con Entregas antes que Recogidas).
- **TSPMS:** Traveling Salesman Problem with Multiple Stacks (Problema del Viajante con Múltiples Pilas).
- **TSPPD:** Traveling Salesman Problem with Pickup and Delivery (Problema del Viajante con Recogida y Entrega de Mercancías).
- **TSPPDL:** Traveling Salesman Problem with Pickup and Delivery and LIFO Loading (Problema del Viajante con Recogida y Entrega de Mercancías y orden LIFO).
- **VND:** Variable Neighborhood Descent (Búsqueda en Entorno Variable Descendente).
- **VNDS:** Variable Neighborhood Decomposition Search (Búsqueda en Entorno Variable con Descomposición).
- **VNS:** Variable Neighborhood Search (Búsqueda en Entorno Variable).
- **VNSMS:** Variable Neighborhood Search with Multi Start (Búsqueda en Entorno Variable con Reinicio).
- **VRP:** Vehicle Routing Problem (Problema de Rutas de Vehículos).
- **VRPB:** Vehicle Routing Problem with Backhauls (Problema de Rutas de Vehículos con Entregas antes que Recogidas).
- **VRPPD:** Vehicle Routing Problem with Pickup and Delivery (Problema de Rutas de Vehículos con Recogida y Entrega de mercancías).
- **VRPTW:** Vehicle Routing Problem with Time Windows (Problema de Rutas de Vehículos con Ventanas de Tiempo).

Índice de figuras

2.1. Una pareja de rutas factibles para un DTSPMS con 6 encargos 39

2.2. Una asignación de pilas factible para un DTSPMS con 6 encargos 40

4.1. Un movimiento de Intercambio en Ruta 84

4.2. Un movimiento de Intercambio Entre Pilas 89

4.3. Un movimiento de Intercambio Dentro de una Pila 93

4.4. Un movimiento de Reinserción 100

4.5. Un movimiento de r-Permutación en Ruta 107

4.6. Un movimiento de r-Permutación en Pila 111

4.7. División de una pila para la realización de un movimiento r-CSP 117

4.8. Un movimiento de r-Permutación Completa en Pila 118

5.1. Reorganización por niveles (ξ_1, ξ_2) del Ejemplo 3 143

5.2. Reorganización global ξ del Ejemplo 3 143

5.3. Representación de la solución 2-factible S de *i)* 146

5.4. Reorganización de las pilas de S según (ξ_1, ξ_2) 146

5.5. Representación de la solución 2-factible S de de *ii)* 147

5.6. Reorganización de las pilas de S según ξ 147

5.7. Reorganización de las pilas de S según ξ_1 148

5.8. Relaciones de contenido entre los algoritmos de reducción de pilas 149

5.9. Distintos planes de carga de la solución S del Ejemplo 4 154

5.10. Planes de carga de la solución potencial del Ejemplo 5 156

5.11. Solución $\{1,6\}$-parcial $\tilde{S}$ del Ejemplo 6 158

5.12. Representación de la solución S del Ejemplo 13 167

5.13. Construcción de $\breve{\Pi}_2(S)=\dot{\Upsilon}_7 \circ \dot{\Upsilon}_4 \circ \Upsilon_7 \circ \Upsilon_4(S)$ 168

5.14. Representación de la solución S del Ejemplo 14 169

5.15. Representación de la solución S_1 del Ejemplo 14 170

5.16. Representación de la solución S_2 del Ejemplo 14 170

5.17. Representaciones de las soluciones S_3 y $\breve{\Pi}_1(S)$ del Ejemplo 14 171

7.1. Resultados de calibración: distinto número de soluciones iniciales 197

7.2. Resultados de calibración: Porcentaje de tiempo dedicado a la intensificación 198

7.3. Resultados de calibración: parámetro *shake*=0,1,2,3 200

7.4. HVNS y GVNS con soluciones α-factibles, $\alpha = 0,1,2,3$ (instancias 33 encargos) 206

7.5. HVNS y GVNS con soluciones α-factibles, $\alpha = 0,1,2,3$ (instancias 66 encargos) 208

7.6. HVNS y GVNS con soluciones α-factibles, $\alpha = 0,1,2,3$ (instancias 132 encargos) 210

7.7. Comparación de algoritmos HVNS y EXT (instancias 33, 66 y 132 encargos) 215

7.8. Comparación de algoritmos HVNS y EHVNS (instancias 33, 66 y 132 encargos) 217

Índice de tablas

4.1. Notación en un movimiento r-CSP 117

7.1. LNS con 33 encargos 194
7.2. LNS con 66 encargos 194
7.3. CCA con 12 encargos y 2 pilas 195
7.4. CCA con 18 encargos y 3 pilas 195
7.5. VNS con soluciones 0-factibles (instancias 33 encargos) 203
7.6. VNS con soluciones 0-factibles (instancias 66 encargos) 204
7.7. VNS con soluciones 0-factibles (instancias 132 encargos) 205
7.8. VNS con soluciones 2-factibles (instancias 33 encargos) 207
7.9. VNS con soluciones 2-factibles (instancias 66 encargos) 209
7.10. VNS con soluciones 2-factibles (instancias 132 encargos) 210
7.11. Mejores resultados con VNS (HVNS) frente a mejores de la literatura (LNS) 211
7.12. Impacto de las estructuras de entornos en el algoritmo HVNS 212
7.13. Resultados de la HVNS con sólo 2 estructuras de entornos 213
7.14. Resultados del algoritmo Exterior (instancias 33, 66 y 132 encargos) 214
7.15. Resultados de la EHVNS (instancias 33, 66 y 132 encargos) 216
7.16. Comparación de los mejores resultados (instancias 33 encargos) 218
7.17. Comparación de los mejores resultados (instancias 66 encargos) 219
7.18. Mejores soluciones obtenidas por HVNS y EHVNS (instancias 132 encargos) 220

7.19. Comparación entre SA y SAR 221