Universidad
Rey Juan Carlos

TESIS DOCTORAL

Un enfoque matheurístico para la resolución del problema de los caminos disjuntos

Autor:

Bernardo Manuel Martín Navas

Directores:

Abraham Duarte Muñoz
Ángel Sánchez Calle

Programa de Doctorado en Tecnologías de la Información y las Comunicaciones

Escuela Internacional de Doctorado

2018

El Dr. D. Abraham Duarte Muñoz, Catedrático de Universidad del Departamento de Ciencias de la Computación, Arquitectura de Computadores, Lenguajes y Sistemas Informáticos y Estadística e Investigación Operativa de la Universidad Rey Juan Carlos, y el Dr. D. Ángel Snchez Calle, Catedrático de Universidad del Departamento de Ciencias de la Computación, Arquitectura de Computadores, Lenguajes y Sistemas Informáticos y Estadística e Investigación Operativa de la Universidad Rey Juan Carlos, directores de la Tesis Doctoral "*Un enfoque matheurístico para la resolución del problema de los caminos disjuntos*" realizada por el doctorando D. Bernardo Manuel Martín Navas,

HACEN CONSTAR:

que esta Tesis Doctoral reúne los requisitos necesarios para su defensa y aprobación.

En Móstoles, a 22 de Mayo de 2018.

Dr. D. Abraham Duarte Muñoz

Dr. D. Ángel Sánchez Calle

A Juan y Olalla

Agradecimientos

Mi más sincero agradecimiento a los directores de la tesis Abraham Duarte y Ángel Sánchez, por darme la oportunidad de trabajar con vosotros, y por toda vuestra confianza, motivación, paciencia y dedicación. Sin vuestro apoyo, hubiera resultado imposible haber finalizado este trabajo de investigación.

Igualmente, querría agradecer a César Beltrán su inestimable ayuda, sus consejos y sugerencias en la investigación. Sus aportaciones resultaron fundamentales para poder llevar las investigaciones por el buen camino.

José Francisco Vélez y Jesús Sánchez-Oro también me ayudaron en diversos momentos en la consecución de esta investigación. También les expreso mi agradecimiento más cordial.

Asímismo debo dar las gracias a los coordinadores del Programa de Doctorado en Tecnologías de la Información y las Comunicaciones José Luis Rojo y Juan José Pantrigo, por las gestiones que han realizado durante mis estudios de doctorado a lo largo de todos estos años.

Este agradecimiento lo hago extensible a Carlos Cuesta por la información suministrada y las gestiones realizadas en relación a esta tesis.

Por otra parte, no me podría olvidar de mis padres Ángel y Manuela, mis hermanos Juan y Ángel, así como del resto de mi familia de Andalucía, Extremadura, Galicia, Madrid, País Vasco y Alemania. Esta gratitud también va dirigida a mis amigos, y a todas las personas que de alguna u otra manera me ayudaron en esta ardua tarea. Gracias a todos por vuestro apoyo constante.

Y por último, gracias Emma por tu paciencia y comprensión.

Índice general

Lista de acrónimos **XV**

Índice de figuras **XX**

Índice de tablas **XXIII**

Índice de algoritmos **XXV**

Resumen **XXVII**

Abstract **XXIX**

1 Introducción **1**
1.1 Un poco de historia . . . 1
1.2 Motivación . . . 3
1.3 Objetivos . . . 4
1.4 Organización del documento . . . 4

2 Marco Conceptual **7**
2.1 Introducción . . . 7
2.2 Complejidad computacional . . . 8
2.2.1 Eficiencia y complejidad . . . 8
2.2.2 Clasificación de problemas . . . 10
2.2.3 Problemas de decisión, lenguajes formales y máquinas de Turing . . . 11
2.2.4 Clases de complejidad . . . 12
2.3 Problemas de optimización . . . 15
2.3.1 Modelos matemáticos . . . 15
2.3.2 Problemas de optimización combinatoria . . . 15
2.3.3 Programación lineal . . . 16
2.3.4 Programación entera . . . 19

3 Métodos de resolución **21**
3.1 Introducción . . . 21
3.2 Métodos Exactos . . . 22
3.2.1 Método simplex . . . 22
3.2.2 Método del punto interior . . . 25

3.2.3 Algoritmo por retroceso . . . 26
3.2.4 Ramificación y acotación . . . 27
3.2.5 Método de planos de corte . . . 30
3.2.6 Ramificación y corte . . . 31
3.2.7 *Branch & Price* . . . 33
3.3 Metaheurísticas . . . 35
3.3.1 Heurísticas . . . 35
3.3.2 Metaheurísticas . . . 36
3.3.3 Intensificación y diversificación . . . 38
3.3.4 Metaheurísticas trayectoriales . . . 39
3.3.5 Metaheurísticas poblacionales . . . 43
3.3.6 Comparación de metaheurísticas . . . 50
3.4 Matheurísticas . . . 51
3.4.1 Introducción . . . 51
3.4.2 Combinación de métodos exactos y búsqueda local como metaheurístico . 52
3.4.3 Combinación de métodos exactos y metaheurísticos: organización colaborativa e integradora . . . 54
3.4.4 Combinación de métodos exactos y metaheurísticos: categorias alto/bajo nivel y *relay*/trabajo en equipo . . . 58
3.4.5 Conclusiones . . . 67

4 Planteamiento del problema 69
4.1 Introducción . . . 69
4.2 El problema de los caminos disjuntos . . . 69
4.2.1 Problemas relacionados . . . 70
4.2.2 Aplicaciones . . . 71
4.2.3 Complejidad del problema . . . 72
4.2.4 Resoluciones previas del problema . . . 78
4.3 Especificación del problema . . . 80
4.3.1 Hipótesis . . . 84
4.3.2 Objetivos . . . 85
4.3.3 Método de investigación . . . 86

5 Método de resolución 89
5.1 Introducción . . . 89
5.2 Una visión global . . . 90
5.3 El modelo de programación entera . . . 91
5.3.1 Relajación del problema . . . 91
5.3.2 Formulación del modelo ILP . . . 92
5.3.3 Implementación del modelo . . . 98
5.4 Metaheurística poblacional . . . 99
5.4.1 Introducción . . . 99
5.4.2 El sistema multiagente . . . 100

5.4.3 Algoritmo de los λ-caminos más cortos . . . 101
5.4.4 Inicialización del agente . . . 103
5.4.5 Fase iterativa del agente . . . 105
5.5 Arquitectura general del método matheurístico . . . 111
5.5.1 Implementación del método metaheurístico . . . 113

6 Resultados **117**
6.1 Introducción . . . 117
6.2 Configuración de experimentos . . . 117
6.2.1 Entorno de despliegue . . . 118
6.2.2 Experimentos e instancias . . . 119
6.2.3 Grafos en las instancias . . . 122
6.2.4 Pares terminales en las instancias . . . 122
6.2.5 Configuración de los parámetros de algoritmo . . . 124
6.3 Ejecución y valoración de los experimentos . . . 128
6.3.1 Comparación del proceso global con cada uno de sus componentes . . . 128
6.3.2 Comparación del matheurístico con los métodos del estado del arte . . . 130
6.3.3 Conclusiones de los experimentos . . . 136

7 Conclusiones y líneas futuras de investigación **137**
7.1 Introducción . . . 137
7.2 Conclusiones . . . 137
7.3 Nuevas ideas de desarrollo . . . 140
7.3.1 Problemas relacionados . . . 140
7.3.2 Métodos de resolución . . . 141

Apéndices **146**

A Resultados detallados por experimentos **147**
A.1 Introducción . . . 147
A.2 Resultados . . . 147
A.2.1 Primera batería de pruebas . . . 148
A.2.2 Segunda batería de pruebas . . . 161

B Contribuciones de la investigación **167**
B.1 Introducción . . . 167
B.2 Aportaciones . . . 167

Bibliografía **169**

Lista de acrónimos

λ-SP *λ-Shortest Paths*. Algoritmo de los λ-caminos más cortos.

ACO *Ant Colony Optimization*. Algoritmo de la colonia de hormigas.

ANTS *Approximate Nondeterministic Tree Search algorithm*.

AS *Ant System*. Sistema de hormigas.

ATSP *Asymmetric Travelling Salesman Problem*. Versión asimétrica del problema del viajante.

B&B *Branch & Bound*. Algoritmo de ramificación y acotación o ramificación y poda.

B&C *Branch & Cut*. Algoritmo de ramificación y corte.

B&P *Branch & Price*

B&P&C *Branch & Price & Cut*

CLK *Chained Lin-Kernighan* (algoritmo específico de tipo ILS).

COP *Combinatorial Optimization Problem*. Problema de optimización combinatoria.

CPU *Central Processing Unit*. Procesador de computador.

CSA *Conformational Space Annealing* (metaheurística).

CVRP *Capacitated Vehicle Routing Problem*. Problema de enrutamiento de vehículos con capacidad.

DCP *Disjoint Connecting Paths*. Rutas de conexión disjuntas. En este caso, se suele referir a caminos con nodos disjuntos.

DEDP *Directed Edge-Disjoint Paths Problem*. Problema de maximización de los caminos con enlaces disjuntos para grafos dirigidos.

EA *Evolutionary Algorithm*. Algoritmo evolutivo.

EC *Evolutionary Computation*. Computación evolutiva.

EDM *Edge-Disjoing Menger Problem*.

EDA *Estimation of Distribution Algorithms*. Algoritmos de estimación de distribuciones.

EDP *Edge-Disjoint Paths Problem.* Problema de maximización de los caminos con enlaces disjuntos.

EDPwC *Edge-Disjoint Paths with Congestion.*

FIFO *First In, First Out* (estructura de datos).

GA *Genetic Algorithm.* Algoritmo genético.

GLB *Gilmore-Lawler Lower Bound Algorithm.*

GP *Genetic Programming.* Programación genética.

GSP *Generalized Steiner Problem.* Problema de Steiner generalizado.

HRH *High-level Relay Hybrid* (categoría de matheurística).

HTH *High-level Teamwork Hybrid* (categoría de matheurística).

IH *Iterated Helsgaun* (algoritmo de resolución metheurístico).

ILP *Integer Linear Programming.* Programación entera.

ILS *Iterated Local Search.* Búsqueda local iterativa.

IPM *Interior Point Method.* El método del punto interior.

JSSP *Job-Shop Scheduling Problem.*

LAP *Linear Assignment Problem.*

LP *Linear Programming.* Programación lineal.

LR *Linear Relaxation.* Relajación lineal.

LRH *Low-level Relay Hybrid* (categoría de matheurística).

LTH (*Low-level Teamwork Hybrid* (categoría de matheurística).

LS *Local Search.* Búsqueda local.

MCKP *Multiconstrained Knapsack Problem.* Problema de la mochila con múltiples restricciones.

MDCP *Maximum Disjoint Connecting Paths.* Problema de maximización de los caminos disjuntos. En este caso, se suele referir a caminos con enlaces disjuntos.

MDKP *Multidimensional Knapsack Problem.* Problema de la mochila multidimensional.

MDPP *Maximum Disjoint Paths Problem.* Problema de maximización de los caminos disjuntos.

MEDP *Maximum Edge-Disjoint Paths Problem.* Problema de maximización de los caminos con enlaces disjuntos.

MH *Metaheuristic*. Algoritmo metaheurístico.

MILP *Mixed Integer Linear Program*. Programación entera-mixta.

MIP *Mixed Integer Linear Program*. Programación entera-mixta.

MINTO *Mixed INTeger Optimizer* (herramienta SW de optimización).

MIQP *Mixed Integer Quadratic Constraint Program*. Programación cuadrática entera-mixta con restricciones.

MIQP *Mixed Integer Quadratic Program*. Programación cuadrática entera-mixta.

MPC *Multi-commodity Flow Problem*.

MSGA *Multi-start Simple Greedy Algorithm*. Algoritmo voraz multiarranque.

NTD Máquina de Turing determinista.

NTND Máquina de Turing no determinista.

NP *Nondeterministic polynomial time*. Tiempo polinomial no determinista (o no determinista en tiempo polinómico) (clase de complejidad).

NPC Clase de complejidad NP-completo.

OE Operación elemental.

P *Polynomial time*. Tiempo polinómico (clase de complejidad).

P2P *Peer-to-Peer*. Red entre pares.

PBIL *Population-Based Incremental Learning*.

PLSP *Production-line Scheduling Problem* (problema de planificación).

PSO *Particle Swarm Optimization*. Algoritmo de enjambre de partículas.

QCP *Quadratic Constraint Program*. Programa cuadrático con restricciones.

QP *Quadratic Program*. Programa cuadrático.

RWA *Routing and Wavelength Assignment Problem*.

SA *Simulated Annealing*. Recocido simulado.

SWO *Squeaky Wheel Optimization* (algoritmo heurístico).

TS *Tabu Search*. Búsqueda tabú.

TSP *Travelling Salesman Problem*. Problema del viajante.

VDP *Vertex-Disjoint Connecting Paths Problem*. Problema de maximización de los caminos con nodos disjuntos.

VLSI *Very Large Scale Integration*. Integración a gran escala.

VLSN *Very Large Scale Neighborhood*. Búsqueda local a gran escala.

VNS *Variable Neighborhood Search*. Búsqueda de vecindad variable.

VRP *Vehicle Routing Problem*. Problema de enrutamiento de vehículos.

VRPTW *Vehicle Routing Problem with Time Windows*. Problema de enrutado de vehículos con ventanas de tiempo.

Índice de figuras

2.1 Diagrama de Euler para los problemas P, NP, NPC y NP-difícil. 14
2.2 Optimización sin restricciones. 17
2.3 Región factible de un problema lineal. 19
2.4 Región factible con una programación entera. 20

3.1 Árbol de expansión con Backtracking. 26
3.2 Progreso de un algoritmo de ramificación y acotación con relajación inicial. . . 29
3.3 Aplicación del método de planos de corte a un modelo de dos variables. 30
3.4 Progreso de un algoritmo de ramificación y corte. 32
3.5 Esquema del algoritmo Branch and Price. 35
3.6 Intensificación y diversificación en un espacio de búsqueda de un problema de optimización. 39
3.7 Ejemplo de búsqueda local iterativa. 42
3.8 Operación de cruce en un GA. 46
3.9 Operación de mutación en un GA. 47
3.10 Representación gráfica del movimiento de una partícula. 49
3.11 Clasificación general de las Matheurísticas. 54
3.12 Hibridación en la fase de diseño. 59
3.13 Cooperación LRH. 60
3.14 Cooperación LTH. 61
3.15 Algoritmo Genético Híbrido LTH. 61
3.16 Organización HRH. 62
3.17 Organización HTH. 63

4.1 Solución del problema EDP para una instancia. 70
4.2 Subdivisión y homeomorfismo de grafos. 75

5.1 Estructura general del algoritmo híbrido a desarrollar. 90
5.2 Cálculo de la función objetivo del modelo. 94
5.3 Restricción en los nodos origen. 94
5.4 Ciclos en los nodos origen. 94
5.5 Nodo origen desempeñando labor de intermediario. 95
5.6 Restricción en nodos intermedios. El flujo tiene que ser cero. 95
5.7 Restricción en nodos intermedios. Casos no contemplados. 95
5.8 Restricción en los nodos destino. 96

5.9 Ciclos en los nodos destino. 96
5.10 Nodo destino desempeñando labor de intermediario. 96
5.11 Capacidad de los enlaces del grafo. 97
5.12 Arquitectura del modelo. 99
5.13 Algoritmo multiagente iterativo: estructura general. 100
5.14 Esquema del agente en una iteración. 101
5.15 Entradas y salidas de un agente en una iteración 1. 104
5.16 Entradas y salidas de un agente en una iteración. 106
5.17 Combinación de soluciones, empleando las ecuaciones 5.15, 5.16, 5.17, 5.18 y 5.19. 108
5.18 Fase iterativa para el agente m. 110
5.19 Arquitectura general del método de dos etapas ILP + sistema multiagente planteado. 112

6.1 Diagrama de contexto del método. 119
6.2 Relación entre experimentos (pruebas o ensayos) e instancias. 120
6.3 Entradas y salidas del experimento. 121
6.4 Análisis de parámetros de configuración realizado por iRace. 128
6.5 Comparación del modelo ILP con el matheurístico. En los ejes se representan el número de pares conectados por cada uno de los métodos. 129
6.6 Comparación del método poblacional con el matheurístico. En los ejes se representan el número de pares conectados por cada uno de los métodos. 130

Índice de tablas

2.1 Comparación de algoritmos polinomiales y exponenciales. 10
2.2 Efectos de la mejora de la tecnología de los computadores sobre los algoritmos polinomiales y exponenciales. 11
2.3 Conjuntos $O(f)$ más importantes y su ordenación. 11

4.1 Relación de los aspectos generales a optimizar en los problemas según el estudio descrito en [8]. 81
4.2 Subcategorías de optimización. 82
4.3 Dominios de los problemas. 82
4.4 Dimensión del problema. 82
4.5 Validación de los resultados de las soluciones desarrolladas. 83
4.6 Asignación de estrategias de optimización a los diferentes problemas. 83

5.1 Notación de la formulación ILP. 93

6.1 Descripción de los grafos de los *benchmarks*. 123
6.2 Descripción de las instancias de los experimentos para la comparativa del matheurístico y los procedimientos encontrados en [34] y [120]. 124
6.3 Descripción de las instancias de los experimentos con la familia de grafos bl-wr2-wht2.10.50 para la comparativa del matheurístico y los procedimientos de [80] y [10]. 124
6.4 Descripción de las instancias de los experimentos con la familia de grafos *mesh* para la comparativa del matheurístico y los procedimientos de [80] y [10]. . . . 125
6.5 Descripción de las instancias de los experimentos con la familia de grafos planos para la comparativa del matheurístico y los procedimientos de [80] y [10]. . . . 125
6.6 Descripción de las instancias de los experimentos con la familia de grafos *steinb* para la comparativa del matheurístico y los procedimientos de [80] y [10]. . . . 126
6.7 Descripción de las instancias de los experimentos con la familia de grafos *steinc* para la comparativa del matheurístico y los procedimientos de [80] y [10]. . . . 126
6.8 Configuración de parámetros. 127
6.9 Mejores configuraciones alcanzadas con la herramienta iRace. 127
6.10 Cualificación de las CPUs empleadas en los experimentos. 131
6.11 Resultados de rendimiento de los algoritmos MSGA, ACO, GA, ILP, metaheurística cooperativa y matheurística. 133

6.12 Resultados de rendimiento de los algoritmos ACO, LS-SGA, LS-R, MP, MP_*reinf*, MSG, ILP, *Metah.* y *Math.* 135

A.1 Descripción de los resultados detallados por cada instancia, y valores promedios para el experimento con el grafo `graph3` y 16 pares terminales de conexión. . . 149
A.2 Descripción de los resultados detallados por cada instancia, y valores promedios para el experimento con el grafo `graph3` y 41 pares terminales de conexión. . . 150
A.3 Descripción de los resultados detallados por cada instancia, y valores promedios para el experimento con el grafo `graph3` y 65 pares terminales de conexión. . . 151
A.4 Descripción de los resultados detallados por cada instancia, y valores promedios para el experimento con el grafo `graph4` y 43 pares terminales de conexión. . . 152
A.5 Descripción de los resultados detallados por cada instancia, y valores promedios para el experimento con el grafo `AS.BA.R-Wax.v100e217` y 10 pares terminales de conexión. 153
A.6 Descripción de los resultados detallados por cada instancia, y valores promedios para el experimento con el grafo `AS.BA.R-Wax.v100e217` y 25 pares terminales de conexión. 154
A.7 Descripción de los resultados detallados por cada instancia, y valores promedios para el experimento con el grafo `AS.BA.R-Wax.v100e217` y 40 pares terminales de conexión. 155
A.8 Descripción de los resultados detallados por cada instancia, y valores promedios para el experimento con el grafo `bl-wr2-wht2.10.50.sdeg` y 50 pares terminales de conexión. 156
A.9 Descripción de los resultados detallados por cada instancia, y valores promedios para el experimento con el grafo `bl-wr2-wht2.10.50.sdeg` y 125 pares terminales de conexión. 157
A.10 Descripción de los resultados detallados por cada instancia, y valores promedios para el experimento con el grafo `bl-wr2-wht2.10.50.sdeg` y 200 pares terminales de conexión. 158
A.11 Descripción de los resultados detallados por cada instancia, y valores promedios para el experimento con el grafo `mesh.25x25` y 62 pares terminales de conexión. 159
A.12 Descripción de los resultados detallados por cada instancia, y valores promedios para el experimento con el grafo `mesh.25x25` y 156 pares terminales de conexión. 160
A.13 Descripción de los resultados detallados por cada instancia, y valores promedios para el experimento con el grafo `mesh.25x25` y 250 pares terminales de conexión. 161
A.14 Descripción de los resultados detallados por cada experimento con el grafo `bl-wr2-wht2.10-50.rand`. 162
A.15 Descripción de los resultados detallados por cada experimento con el grafo `bl-wr2-wht2.10-50.sdeg`. 162
A.16 Descripción de los resultados detallados por cada experimento con el grafo `mesh15x15`. 163
A.17 Descripción de los resultados detallados por cada experimento con el grafo `mesh25x25`. 163

A.18 Descripción de los resultados detallados por cada experimento con el grafo `steinb10`. 164
A.19 Descripción de los resultados detallados por cada experimento con el grafo `steinb16`. 164
A.20 Descripción de los resultados detallados por cada experimento con el grafo `steinc6`. 164
A.21 Descripción de los resultados detallados por cada experimento con el grafo `steinc11`. 165
A.22 Descripción de los resultados detallados por cada experimento con el grafo `steinc16`. 165
A.23 Descripción de los resultados detallados por cada experimento con el grafo `planar.50`. 165
A.24 Descripción de los resultados detallados por cada experimento con el grafo `planar.100`. 166
A.25 Descripción de los resultados detallados por cada experimento con el grafo `planar.200`. 166
A.26 Descripción de los resultados detallados por cada experimento con el grafo `planar.500`. 166

Índice de algoritmos

3.1 Backtracking 26
3.2 Cutting Plane 30
3.3 Branch & Cut 32
3.4 Simulated Annealing 40
3.5 Tabu Search 41
3.6 Iterated Local Search 41
3.7 Multi-start Simple Greedy Algorithm 43
3.8 Ant System 44
3.9 Evolutionary algorithm 45
3.10 Genetic Algorithm 45
3.11 Particle Swarm Optimization 50

4.1 ρ-Greedy Algorithm 77

5.1 λ-Shortest Paths 102
5.2 EDP: matheuristic 113
5.3 EDP: metaheuristic 114
5.4 EDP MH: initIndividual 114
5.5 EDP MH: combinationStep 115
5.6 EDP MH: localSearch 115

Resumen

El problema de los caminos disjuntos (*Edge-disjoint Paths Problem*) consiste en maximizar el número de pares terminales conectados en una red a través de caminos con arcos disjuntos. Éste es un problema clásico NP-completo con amplias aplicaciones en diversas áreas, como pueden ser las telecomunicaciones o el diseño de circuitos integrados, entre otros.

Entre los métodos de resolución para este tipo de problema de optimización se encuentran los métodos exactos y metaheurísticos, ambos con diversas fortalezas y debilidades asociadas. Para tomar ventaja del uso de estas técnicas se desarrollaron las *matheurísticas*, que pretenden englobar los méritos de ambos métodos mediante un procedimiento híbrido de resolución.

En esta tesis se propone resolver el problema mencionado anteriormente empleando un matheurístico secuencial de dos fases: como primera etapa se desarrolla un modelo exacto de programación entera; para continuar con una metaheurística poblacional, que tiene como objetivo perfeccionar la solución alcanzada por el primer paso.

Este algoritmo híbrido de dos etapas mejora los resultados de los métodos de resolución encontrados en el estado del arte que resuelven el problema planteado. Por un lado, el procedimiento original propuesto en este documento conectará un mayor número de pares en la red; y por otra parte, ejecutará dicha acción consumiendo una menor cantidad de tiempo de cómputo que en el resto de métodos encontrados en la literatura.

Abstract

The Edge-disjoint Paths Problem has the objective of maximizing the number of terminal pairs connected in a network by means of paths with disjoint arcs. This is a classic problem NP-complete with wide applications in diverse areas, such as the telecommunications networks or the design of integrated circuits, among others.

In order to solve optimization problems, we can find exact and metaheuristic resolution methods, both of them with different strengths and associated weaknesses. To take advantage of the use of these techniques, matheuristics were developed, which aim to encompass the merits of both methods by means of a hybrid resolution procedure.

In this thesis, it is proposed a two-phase matheuristic which solves the aforementioned problem: in the first step, an integer linear programming model is developed; in the second one, it is executed a population metaheuristic, which aims to improve the solution reached by the first step.

This two-step hybrid algorithm improves the results calculated by previous resolution methods found in the state of the art. On the one hand, the original procedure proposed in this document connects a greater number of pairs; and on the other hand, it consumes a smaller amount of computing time than the rest of the methods found in the literature.

Capítulo 1

Introducción

1.1. Un poco de historia

Tal y como se comenta en *A Brief History of NP-Completeness, 1954-2012* de David S. Johnson [125], ya en 1956 Kurt Gödel escribió una carta a John von Neumann, ambos del Instituto de Estudios Avanzados de Princeton, en los que se trataba la NP-completitud [1] en los problemas. En esa época, el primero ya utilizaba la idea del "tiempo de ejecución en el peor de los casos"; algo que, si bien ahora es un concepto bastante común, en aquel tiempo no lo era tanto. Parece que ya tenía en mente la noción de clases de problemas resolubles mediante una búsqueda exhaustiva, lo cual podría ser visto como una generalización del concepto NP. Asímismo, lanzaba una pregunta final precursora de la cuestión más común ¿$P = NP$?

El famoso matemático John Nash también fue un precursor de la teoría de la NP-completitud. Admitía que no podía demostrar esa conjetura, ni esperaba que fuera probada, aunque fuese verdad.

La teoría de la NP-completitud se remonta a Steve Cook en 1971, con el trabajo *The complexity of theorem-proving procedures* [62], que contenía los primeros resultados publicados de la NP-completitud. Sin embargo, Leonid Levin, entonces un estudiante en Moscú, demostró los mismos resultados casi al mismo tiempo, aunque no fuesen publicados hasta 1973. Con los años, lo que solía llamarse "Teorema de Cook", ahora se conoce generalmente como el "Teorema de Cook-Levin".

El artículo de Cook fue reconocido por su importancia tan pronto como apareció. No sólo demostró que la "satisfabilidad" es NP-completo (en la terminología moderna), sino que también demostró lo mismo para el problema $3SAT$. Además, Cook insinuó una aplicación más amplia del concepto, mostrando que lo mismo ocurre con el problema de isomorfismo de subgrafos (más específicamente, el caso especial ahora conocido como el problema de "clique").

[1] La clase de problemas resolubles en tiempo polinómico se suelen denotar por P. La clase NP, que se describirá con detalle en el capítulo 2.2 del documento, contiene problemas que "parecen" (muchos cientficos consideran que "lo son") más difíciles de resolver. Esa clase contiene la mayoría de los problemas que tienen interés en la optimización combinatoria, incluyendo lógicamente los problemas que están en P [78]. NP-completo es el subconjunto de los problemas de decisión en NP de tal forma que todo problema incluido en NP se puede reducir a cada uno de los problemas que pertenecen a la clase de los NP-completo. Se puede decir que los problemas integrados en el conjunto NP-completo son los problemas más difciles de NP.

Richard Karp publicó en 1972 el artículo *Reducibility among combinatorial problems* [130]. Este documento demostraba la NP-completitud de 19 problemas adicionales, entre los que se encuentran los problemas de cobertura de vértices (*Vertex Cover*), del número cromático (*Chromatic Number*), del circuito hamiltoniano directo e indirecto, de la suma de subconjuntos (*Subset Sum*) y el problema de la mochila (*Knapsack Problem*). También parece ser que fue el primero en usar las anotaciones P y NP, aunque su término para NP-completo fuera "polinomial completo", una locución utilizada en varios trabajos anteriores a que la terminología moderna tomara posesión. El trabajo también introdujo la distinción entre una "transformación polinómica", donde una instancia de un problema se transforma en otro que tiene la misma respuesta "sí-no"; y una "reducción polinómica", en la que el primer problema se resuelve usando uno o más llamadas a una subrutina.

Por otro lado, Levi describe lo que ahora se vería como la noción estándar de una reducción polinómica de un problema de búsqueda a otro [154]. Y considera que un problema es un "problema de búsqueda universal", si existen reducciones polinómicas a ésta desde todos los ejercicios de búsqueda que pertenecen a la clase del problema anterior. Asímismo, demuestra que un conjunto (de seis elementos) de problemas son de búsqueda universal, entre los que se encuentran la satisfacibilidad, la cobertura de vértices y el isomorfismo de subgrafos.

En 1979 aparece el libro *Computers and Intractability: A Guide to the Theory of NP-completeness* escrito por Michael Garey y David Johnson [99]. En este documento se ocupan de más de 300 problemas NP-completos, entre los que se encuentra el ejercicio ND40 de conexión de rutas disjuntas (*Disjoint Connecting Paths*). Es de tipo NP, debido a que se puede "reducir" a él desde el problema clásico NPC 3-SAT de Karp [139], [129], [158] y [130].

El problema de conexión de rutas disjuntas básicamente indica que, teniendo un grafo $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, con $\mathcal{V}$ los vértices y $\mathcal{E}$ los enlaces de la estructura; y una colección de pares de nodos disjuntos $(s_1, t_1), (s_2, t_2), ..., (s_k, t_k)$, se pregunta si $\mathcal{G}$ contiene k rutas con vértices disjuntos, cada cual conectando s_i a t_i, para cada i, $1 \leq i \leq k$.

Este ejercicio es similar al problema de maximización de caminos disjuntos (*Maximum Disjoint Connecting Paths*) MDCP tal y como aparece en la recopilación de problemas NP en [68]; y consiste en el siguiente enunciado:

Sea un grafo $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ y un conjunto de pares de vértices $\mathcal{T} = \{(s_1, t_1), (s_2, t_2), ..., (s_k, t_k)\}$, se desea maximizar el número de rutas con enlaces disjuntos en $\mathcal{G}$ que conecten los pares de $\mathcal{T}$.

El documento de Johnson [125] continúa en su cronología con la resolución de diversos problemas NP-completos mediante los algoritmos de aproximación; pero, en este caso, ya se ha encontrado el objeto de las investigaciones sobre las que versará este documento.

Más exactamente, se trata de resolver, utilizando algunas de las técnicas de resolución de problemas NP, el último problema mencionado en estas líneas: el problema de maximización de caminos con enlaces disjuntos MDCP; también nombrado como EDP (*Edge-Disjoint Paths Problem*) o MEDP (*Maximum Edge-Disjoint Paths Problem*). Se deberán aportar soluciones originales y de calidad, que superen a las actuales aparecidas en el estado del arte, y con tiempos de cómputo competitivos.

1.2. Motivación

La ciencia de la computación debe tener como uno de sus objetivos, el desarrollo de soluciones que sean capaces de resolver problemas de una forma eficiente. Para ello, deberá aportar procesos adecuados que permitan obtener los mejores resultados, mitigando la cantidad de recursos *hardware* y *software* necesarios para su implementación.

Cuando los problemas a resolver son de tipo NP, implica un mayor desafío, debido a la naturaleza del ejercicio. Es necesario elegir cuidadosamente el método encargado de llevar a cabo la resolución porque, en otro caso, los resultados no serán los esperados. Para ello, además de conocer el problema, es imprescindible tener una gran comprensión del abanico de técnicas y métodos de resolución disponibles. Esto es importante porque además se debe tener pericia en la adaptación del problema al procedimiento.

De entre las diferentes técnicas que se encuentran en el estado del arte y que resuelven los problemas de optimización, hay dos que se van a destacar. Por un lado, se encuentran los métodos exactos. Estos procesos que, por una parte, se suelen basar de una manera primordial en las matemáticas; y por otra, acostumbran a realizar procesos exhaustivos de búsqueda, pueden asegurar los resultados óptimos del problema. Pero si éste es de tipo NP, y las instancias o entradas de datos empiezan a aumentar en tamaño, el ejercicio se puede convertir en intratable para los métodos anteriores, debido principalmente al tiempo de ejecución y a la memoria necesaria para llevar a cabo dicho proceso [79].

Por otro lado, existen otras dos técnicas fundamentales para abordar los problemas de optimización. Éstas son las heurísticas y las metaheurísticas. Los primeros son métodos [97], [234] basados fundamentalmente en la intuición y en el sentido común, alejándose del formalismo de las matemáticas de los métodos exactos. El inconveniente de estos procedimientos se debe a que tienden a alcanzar óptimos locales como resultados. Para evitar estas situaciones, éstas se suelen complementar con las metaheurísticas, que se pueden considerar como "plantillas" diseñadas para evitar los máximos y mínimos locales de los procedimientos heurísticos. La combinación de ambos procesos suele ofrecer soluciones cercanas a las óptimas, implementándose de una manera más fácil y flexible que empleando los modelos exactos [35], [96].

Con todo lo anterior, se podría intuir que una colaboración entre algoritmos exactos y heurísticos/metaheurísticos podrían proporcionar métodos híbridos que engloben las ventajas y fortalezas de cada uno de ellos. A estos procedimientos mixtos se le llaman matheurísticas.

La utilización de una matheurística en la resolución del problema puede proporcionar datos de su rendimiento, no sólo generales del proceso global, sino también particulares asociados a cada uno de los componentes exactos y heurísticos (o metaheurísticos) que forman parte del proceso. Esta información podría suministrar conclusiones respecto a la diferencia de comportamiento de los dos métodos (exacto y heurístico/metaheurístico) en el problema.

Para finalizar, el problema clásico a resolver tiene multitud de aplicaciones en diferentes áreas tales como las telecomunicaciones o la integración de circuitos integrados (VLSI) [131], [153], [50]. Se han llevado a cabo investigaciones en procesos de enrutamiento a gran escala [20], [191], [119], [10], y en redes que permiten altas velocidades, como pueden ser las de fibra óptica [40], [198], en donde la problemática de establecer redes virtuales viene en gran medida dada

por la obtención de buenas heurísticas que permitan encontrar los caminos disjuntos.

1.3. Objetivos

Como se comentó en los puntos anteriores, esta memoria se centra en la resolución del problema clásico NP-completo de maximización de los caminos disjuntos (*Edge Disjoint Path Problem* o EDP) en un grafo. Con mayor detalle, este ejercicio se puede definir de la siguiente forma [24]:

Dado un grafo $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ con $\mathcal{V}$ los vértices y $\mathcal{E}$ los enlaces de la estructura; y un conjunto $\mathcal{T}$ de solicitudes de conexión entre pares de nodos terminales $\mathcal{T} = \{(s_1, t_1), (s_2, t_2), , (s_k, t_k)\}$ en $\mathcal{V}$, se trata de maximizar el número de duplas (s_i, t_i), para todo $i = 1, 2, ..., k$, que se puedan conectar mediante caminos formados por arcos disjuntos.

La propuesta que se realice en este documento deberá generar mejores resultados, tanto en el número de caminos conectados (función objetivo), como en el tiempo necesario para su consecución, a los desarrollados previamente y que se encuentren en el estado del arte. Para realizar un análisis comparativo con los algoritmos previos, se deberán obtener resultados sobre las mismas instancias que las utilizadas en aquellos procedimientos. De esta forma, el contraste de las soluciones adquiridas en este trabajo con los conseguidos en los métodos anteriores, se podrá realizar de la manera más objetiva.

Por último, es importante tener en cuenta que el procedimiento a desarrollar, además de alcanzar las mejores soluciones, deberá ser original; es decir, no implementado y publicado anteriormente. En este caso, se anticipa que el matheurístico contendrá una formulación entera (*Integer Linear Programming* -ILP-) que se resolverá mediante la herramienta CPLEX; y el metaheurístico será un algoritmo poblacional. Los resultados también se podrían emplear para obtener conclusiones relativas a la diferencia del conducta entre el método exacto y el metaheurístico, incluidos ambos en la proceso híbrido a ejecutar.

1.4. Organización del documento

En los primeros capítulos de la memoria, se realiza una revisión bibliográfica de los diferentes conceptos necesarios para abordar el problema, que se organizan fundamentalmente desde las ideas generales a las particulares. Teniendo en cuenta que se resuelve un problema de optimización NP, el capítulo 2 incluye algunas nociones relativas a los ejercicios de optimización y materias vinculadas con la complejidad de algoritmos.

A continuación, en el capítulo 3 se abordará con más detalle algunos métodos de resolución de problemas. Se incluyen métodos tanto exactos como metaheurísticos. En los apartados incluídos se comentan diferentes técnicas de desarrollo:

- Por una parte, las empleadas en la propia resolución del problema, y que deben lograr el objetivo establecido en este documento.
- Y por otro lado, en general cualquier otro procedimiento que aparezca o se especifique en la memoria.

También se incluye en este capítulo diversas clasificaciones de las matheurísticas según su estructura y tipología. Este contenido es fundamental, porque actuará de guía para determinar aspectos relacionados con la organización, estructura e interacción de los elementos que forman parte del método híbrido a implementar.

En el capítulo 4 se detallan algunos aspectos del problema EDP a resolver, como pueden ser los relativos a su complejidad; así como referencias a los procesos implementados anteriormente que solucionan el problema, y que servirán de referencia para realizar los análisis de rendimiento del método propuesto.

También en este capítulo se detalla el ejercicio a solucionar, incluyendo las hipótesis previas que se presuponen que se cumplirán con la consecución del mismo; así como la definición de los objetivos que se deben satisfacer con el desarrollo de esta investigación.

En el capítulo 5 se detallan los componentes *software* que se utilizarán para resolver el problema propuesto. En este caso, se aportará un método híbrido formado por un modelo ILP y un módulo metaheurístico poblacional.

En el capítulo 6 se incluyen los experimentos que ayudarán a medir las prestaciones del procedimiento desarrollado, contrastando sus soluciones con las alcanzadas mediante diversos métodos encontrados en el estado del arte. De esta manera se podrá comprobar la calidad de los resultado obtenidos mediante el nuevo proceso de resolución.

Las conclusiones más importantes que se pueden deducir con el desarrollo de este trabajo, así como algunas ideas y líneas de investigación, que podrían ser abordadas a raíz de las investigaciones descritas en este documento, se encuentran en el capítulo 7;.

En la parte final de la memoria, se incluyen dos apéndices: el primero dedicado a detallar más los experimentos descritos en el capítulo 6; y el segundo a incluir las contribuciones y artículos realizados a partir de las investigaciones presentadas en esta memoria.

Al final de esta memoria aparece la bibliografia empleada en este trabajo de tesis doctoral.

Capítulo 2

Marco Conceptual

2.1. Introducción

Este capítulo se centra en describir algunos aspectos relativos a la complejidad de algoritmos y a los problemas de optimización.

Tal y como se comenta en la referencia [110], los algoritmos tienen dos puntos muy importantes a tener en cuenta: éstos son el diseño y la eficiencia. El primero se refiere a la búsqueda de la secuencia adecuada de instrucciones que permitan resolver el problema. El segundo, en cambio, se refiere a medir el coste, en tiempo y recursos que utiliza un procedimiento para encontrar una solución. En los siguientes apartados de este capítulo, se expondrá con más detenimiento el segundo concepto.

Por otra parte, la optimización es una disciplina fundamental en campos de la Ciencia de la Computación, la Inteligencia Artificial o la Investigación Operativa [74]. Incluso puede ser vista como piedra angular de las otras [92]. Trata de buscar soluciones factibles a problemas de la vida real, y tiene aplicaciones en Ingeniería, Medicina, Economía y otras muchas áreas científicas [92], [57].

Tal y como se cuenta en [166], los problemas de optimización consisten en obtener un conjunto de valores (o configuración de variables del problema) que satisfagan condiciones intrínsecas de la cuestión a resolver y a la vez cumplan con determinados objetivos.

Estos problemas tratan con la dificultad de encontrarse en general con muchos posibles resultados, y alguna forma clara de comparación entre ellos [77]. La calidad de cada una de dichas soluciones es evaluada por medio de la denominada función objetivo. Por tanto, estos algoritmos resuelven instancias de problemas que se piensan que son difíciles, explorando el espacio de soluciones, usualmente extenso. Para llevar a cabo este proceso, tratan de reducir el tamaño efectivo del espacio de búsqueda, explorándolo eficientemente.

2.2. Complejidad computacional

2.2.1. Eficiencia y complejidad

Una vez que se dispone de un algoritmo que funciona correctamente; es decir, que cumple con el cometido para el que fue creado, se pueden definir criterios que midan su rendimiento. Entre las métricas para valorar este concepto, se pueden encontrar la simplicidad y el uso eficiente de los recursos.

Esta última, se suele calcular en función de dos parámetros: el espacio, es decir, la memoria que consume; y el tiempo que tarda en ejecutarse. Además, esta valoración se puede utilizar para realizar la comparación entre diferentes algoritmos que tengan el mismo propósito, y así determinar cuál es el más adecuado.

Como se indica en [91], no sólo se debe tener en consideración el hecho de que el proceso realice lo especificado, sino que también lo efectúe con los recursos disponibles. Dentro de la eficiencia temporal, el tiempo de ejecución del algoritmo va a depender de diferentes factores como pueden ser los datos de entrada, la calidad del código objeto generado por el compilador, la naturaleza de las instrucciones del procesador que ejecuta el programa y la complejidad intrínseca del procedimiento. Hay dos posibles estudios sobre el tiempo de cómputo:

1. Por un lado, se puede proporcionar una medida teórica, a priori. Consiste en obtener una función que acote el tiempo de ejecución del algoritmo.

2. Y por otra parte, se puede ofrecer una medida real, a posteriori. En este caso, trata de medir el tiempo de ejecución del proceso para unos valores de entrada dados, y para un computador en concreto.

Ambas medidas son importantes puesto que si bien la primera nos ofrece estimaciones del comportamiento del procedimiento de forma independiente del *hardware*, la segunda representa las medidas reales del proceder del algoritmo.

Se entiende por tamaño de la entrada, al número de componentes sobre los que se va a ejecutar el algoritmo. Por ejemplo, la longitud del vector a ordenar o la dimensión de las matrices a multiplicar.

La unidad de tiempo a la que deben hacer referencia el estudio teórico de eficiencia no puede ser expresada en segundos o en ninguna unidad de tiempo concreta, debido a que no existe un computador estándar sobre el que se pueda hacer referencia todas estas magnitudes. Como consecuencia, se denotará como $T(n)$ al tiempo de ejecución de un algoritmo para una entrada de tamaño n.

Teóricamente $T(n)$ debe indicar el número de instrucciones u operaciones básicas ejecutadas por un computador idealizado; siendo ésta una medida simple y abstracta, independiente del *hardware* a utilizar. Para ello es necesario acotar de alguna forma la diferencia que se puede producir entre diferentes implementaciones de un mismo procedimiento: ya sea del mismo código ejecutado en diferentes máquinas; o bien de diversos compilaciones (lenguajes de programación, etc.) que implementen el mismo método. Esta diferencia es la que acota el "principio de invarianza", que se explicará a continuación. Por otra parte $E(n)$ será el espacio de memoria requerido para la ejecución del algoritmo con una entrada de tamaño n.

Principio de invarianza Dado un algoritmo, y dos implementaciones suyas I_1 e I_2, que tardan $T_1(n)$ y $T_2(n)$ segundos respectivamente, el principio de invarianza afirma que existe una constante real $c > 0$ y un número natural n_0 tales que para todo $n \geq n_0$ se verifica que $T_1(n) \leq cT_2(n)$. Es decir, el tiempo de ejecución de dos desarrollos distintos de un algoritmo dado no va a diferir más que en una constante multiplicativa. Con lo anterior, se puede afirmar que un procedimiento tarda un tiempo del orden de $T(n)$ si existe una constante real $c > 0$ y una implementación I del algoritmo que tarda menos de $cT(n)$, para todo n tamaño de la entrada.

A la hora de medir el tiempo, se hará en función del número de "operaciones elementales" (OE) que realice dicho algoritmo, entendiendo por operaciones elementales aquellas que realiza el computador en un tiempo acotado por una constante. De esta forma, se considera como OE a las operaciones aritméticas básicas, asignaciones a variables de tipo predefinido por el compilador, los saltos como llamadas a funciones u otros procedimientos, y el retorno desde ellos. Las comparaciones lógicas y el acceso a estructuras indexadas básicas como vectores o matrices también son aceptadas como OE. Cada una de ellas contabilizará 1 OE.

Resumiendo, se calcula el tiempo de ejecución de un algoritmo mediante una función que mide el número de operaciones elementales que realiza dicho proceso para un tamaño de entrada dado.

Cotas de complejidad. Medidas asintóticas

Una vez visto la forma de calcular el tiempo de ejecución T de un algoritmo, se deben clasificar las funciones obtenidas para una posible comparación. Así, se definen clases de equivalencia correspondientes a las funciones que "crecen de la misma forma".

Cota superior. Notación O Dada una función f, se quieren estudiar aquellas funciones g que a lo sumo crezcan tan deprisa como f. Al conjunto de tales funciones se le llama "cota superior de f ó $O(f)$". Conociendo la cota superior de un algoritmo, se podría asegurar que en ningún caso el tiempo empleado en la ejecución será superior al de la cota.

Sea $f : \mathbb{N} \to [0, \infty)$ Se define el conjunto de funciones de orden O de f como:

$$O(f) \to \{g : \mathbb{N} \to [0, \infty) \mid \exists c \in \mathbb{R},\ c > 0,\ \exists n_0 \in \mathbb{N},\ g(n) \leq cf(n),\ \forall n \geq n_0\} \tag{2.1}$$

Si $t \in O(f)$ indica que t se encuentra acotada superiormente por algún múltiplo de f. Se debería buscar la menor función f tal que $t \in O(f)$.

Cotas asintóticas La utilización de las cotas asintóticas para comparar funciones de tiempo de ejecución, se basa en la hipótesis de que son suficientes para decidir el mejor algoritmo, prescindiendo de las constantes de proporcionalidad. Sin embargo, esta afirmación puede no ser cierta cuando el tamaño de la entrada es pequeño. Dado un algoritmo, se dirá que su orden de complejidad es $O(f)$ si su tiempo de ejecución para el peor caso es de orden O de f, es decir, $T_{peor}(n)$ es de orden $O(f)$.

2.2.2. Clasificación de problemas

El objetivo fundamental de la complejidad computacional es clasificar los problemas de acuerdo a su "tratabilidad", tomando el o los algoritmos más eficientes para resolverlos. Dado un proceso que resuelve un ejercicio, es razonable -y aconsejable- preguntarse si no existirá un procedimiento más eficiente para el mismo problema.

Por tanto, a la hora de abordar una cuestión, se debería responder las siguientes preguntas: ¿Es el problema tratable? Si el problema es tratable, ¿es el algoritmo suficientemente eficiente? Si se demuestra que el ejercicio no admite mejores soluciones, entonces se puede afirmar que el algoritmo es eficiente.

Un algoritmo de tiempo polinomial es aquel que tiene la función de complejidad del orden de $O(p(n))$, donde p es una función polinómica y n denota la longitud de la entrada. Cualquier otro algoritmo cuya función de complejidad no pueda ser limitada por una función polinómica, se denomina algoritmo de tiempo exponencial.

Pruebas empíricas han permitido acordar que, si un problema admite una solución polinomial, entonces se trata de un ejercicio "tratable"; en caso contrario se consideraría "intratable". El límite de tratabilidad entre algoritmos polinomiales y super-polinomiales es arbitrario. Por ejemplo, $1,00001^n$ ó $n^{log_2 n}$ podrían ser intratables; mientras que n^{10000} ó $100000000000 \cdot n$ serían tratables. La diferencia entre los dos tipos de algoritmos (tratable: polinómicos, intratables: exponenciales) toma verdadero significado cuando se aplica sobre instancias de problemas de grandes dimensiones. Esto se puede apreciar en [99], donde se muestra la tabla 2.1 que puede ayudar a tener una mejor idea de la diferencia entre los algoritmos tratables e intratables. En este caso, la función $f(1)$ presenta tiempo de ejecución de $1\mu s$ ó $10^{-6}s$.

	Tamaño n de la entrada					
f	10	20	30	40	50	60
n	0,00001s	0.00002s	0.00003s	0.00004s	0.00005s	0.00006s
n^2	0.0001s	0,0004s	0.0009s	0.0016s	0.0025s	0.0036s
n^3	0.001s	0.008s	0.027s	0.064s	0.125s	0.216s
n^5	0.1s	3.2s	24.3s	1.7 min	5.2 min	13.0 min
2^n	0.001s	1s	17.9 min	12.7 días	35.7 años	366 siglos
3^n	0.059s	58 min	6.5 años	3855 siglos	$2 \cdot 10^8$ siglos	$1{,}3 \cdot 10^{13}$ siglos

Tabla 2.1: Comparación de algoritmos polinomiales y exponenciales.

En la tabla siguiente (también tomada de [99]) se detalla los efectos que tiene la mejora de la tecnología del computador sobre las funciones de complejidad, tomando como base la instancia del problema que se resuelve en una hora.

Como se puede apreciar en la tabla 2.2, la mejora del *hardware* del computador tiene mayores efectos en los algoritmos de complejidad polinomial; mientras que en los exponenciales, apenas aumenta la velocidad de cómputo.

Además de las polinomiales y exponenciales, existen otras funciones de complejidad, como se muestran a continuación (tabla 2.3). Cada orden es subconjunto del siguiente: $O(1) \subset O(log\, n) \subset O(n) \subset O(n\, log\, n) \subset O(n^c) \subset O(c^n) \subset O(n!) \subset O(n^n)$.

f	Computador actual	Computador 100 veces más rápido	Computador 1000 veces más rápido
n	T_1	$100 \cdot T_1$	$1000 \cdot T_1$
n^2	T_2	$10 \cdot T_2$	$31{,}6 \cdot T_2$
n^3	T_3	$4{,}64 \cdot T_3$	$10 \cdot T_3$
n^5	T_4	$2{,}5 \cdot T_4$	$3{,}98 \cdot T_4$
2^n	T_5	$T_5 + 6{,}64$	$T_5 + 9{,}97$
3^n	T_6	$T_6 + 4{,}19$	$T_6 + 6{,}29$

Tabla 2.2: Efectos de la mejora de la tecnología de los computadores sobre los algoritmos polinomiales y exponenciales.

Problema	Órden	Nombre
Tratable	$O(1)$	constante
	$O(log\, n)$	logarítmico
	$O(n)$	lineal
	$O(n^c)$ con $c > 1$	polinómico
No tratable	$O(c^n)$ con $c > 1$	exponencial
	$O(n!)$	factorial
	$O(n^n)$	combinatorio

Tabla 2.3: Conjuntos $O(f)$ más importantes y su ordenación.

2.2.3. Problemas de decisión, lenguajes formales y máquinas de Turing

Problemas de decisión

Los problemas de decisión son aquellos que tienen como respuesta "sí" o "no". Se puede establecer una asociación entre los problemas de decisión y los lenguajes formales, de tal forma que se pueden emplear matemáticamente estos últimos para poder obtener conclusiones y clasificaciones de los primeros. De esta forma, la complejidad de un lenguaje L es la complejidad del problema de decisión asociado a L.

Tesis de Church-Turing

La tesis de Church-Turing formúla hipotéticamente la equivalencia entre los conceptos de función computable y máquina de Turing; que expresado en lenguaje corriente viene a decir "todo algoritmo es equivalente a una máquina de Turing". No es un teorema matemático, sino una afirmación formalmente indemostrable; o una hipótesis que, no obstante, tiene una aceptación prácticamente universal.

Máquina de Turing determinista y no determinista

Mientras que una máquina determinista (MTD) sigue un único "camino computacional", una no determinista (MTND) emplea en su caso un "árbol de cómputo". La MTND determina qué acción tomar entre varias posibles, "clonándose" y bifurcándose en varias copias, cada una de las cuales sigue una de las posibles transiciones. Si cualquiera de las ramas del árbol finaliza en un estado final de aceptación, se dice que la máquina acepta la entrada.

Si la entrada x es aceptada por la MTND en t pasos, entonces x es aceptada por la MTD en c^t pasos, para alguna constante positiva c. La sentencia anterior permite simular las máquinas de Turing no deterministas como máquinas de Turing deterministas, pero con un coste exponencial de tiempo.

Lenguaje aceptable y decidible

Un lenguaje L es "aceptable" si existe una MTD T tal que $L = L(T)$. O lo que es equivalente, la máquina se "para" al reconocer una cadena x del lenguaje. Un lenguaje L es "decidible" si existe una MTD T tal que $L = L(T)$; y además, $\forall x$, alfabeto de entrada a la máquina T, $T(x)$ es finita. Esta propiedad implica que:

- La máquina dice si una cadena pertenece al lenguaje o no.
- Conlleva reconocer el complemento del lenguaje L^c.
- En cualquier caso, la máquina al final finaliza. Es decir, no entra en bucle (aunque $x \notin L$; o lo que es lo mismo, T no encuentre solución para x).

2.2.4. Clases de complejidad

Clase de complejidad P

La clase P contiene a aquellos problemas que son solubles en tiempo polinómico por una máquina de Turing determinista. La clase P juega un papel importante en la teoría de la complejidad computacional debido a que, a grandes rasgos, P corresponde a la clase de problemas que, de manera realista, son solucionables por una computadora (se puede suponer que es un computador como los ordenadores personales clásicos).

Pertenencia a la clase P y "tratabilidad"

En general se está de acuerdo que la clase P representa a los problemas tratables. Sin embargo existen ciertos problemas en P que no son realmente tratables, debido a diferentes causas. Por ejemplo, que el grado k del polinomio sea un número grande.

P se considera como una aproximación razonable al concepto de tratabilidad. Es posible demostrar que un problema pertenece a P de dos formas:

- Obteniendo un algoritmo polinomial.
- Utilizando una "reducción" a otro problema que ya se sabe que se encuentra en P.

Para el segundo caso se utiliza el concepto de "reducibilidad polinomial", de tal forma que transforme, en tiempo polinomial, una instancia del problema expresado como un lenguaje L_1 en una instancia del problema L_2, y así sus respuestas sean las mismas. La reducción también podría plantearse como una transformación de un problema en otro. Intuitivamente, un problema Q puede ser reducido a otro Q', si cualquier instancia del ejercicio Q puede ser "fácilmente" expresada como una instancia de Q'; y cuyo resultado proporcione una solución para la instancia de Q.

Clase de complejidad NP

Para algunos problemas, no se conocen procedimientos que los resuelvan en tiempo polinómico. Quizá estos ejercicios tengan algoritmos en tiempo polinomial que se basan en principios por ahora desconocidos; o es posible que estos problemas no puedan ser resueltos en tiempo polinómico debido a que son "inherentemente difíciles".

La clase de complejidad NP consta de los problemas "verificables" en tiempo polinómico. Por verificable se entiende a un ejercicio tal que dado un candidato a solución, se puede verificar que dicho candidato es o no solución en un tiempo polinómico [99].

El término NP proviene de "no determinista en tiempo polinómico". O lo que es lo mismo, un problema NP es aquel que puede ser resuelto por una máquina de Turing no determinista en tiempo polinómico. Y así parece razonable intuir que para los problemas NP, no se puede esperar soluciones que tengan un rendimiento mejor que exponencial, debido fundamentalmente a que si un único computador (MTD) tiene que probar el número total de soluciones, que puede ser exponencial, el rendimiento total será asímismo exponencial, aunque las soluciones particulares tengan rendimiento polinomial.

La pregunta ¿$P = NP$?

Intuitivamente, se cree que P es un subconjunto de NP. Y efectivamente, cada problema de decisión resuelto por un algoritmo de tiempo polinomial determinista, también puede serlo por uno de tiempo polinomial no determinista. Simplemente se necesita observar que cualquier algoritmo determinista puede ser utilizado en la etapa de verificación de un algoritmo no determinista.

La pregunta $P = NP$ es una de las más importantes en el campo de las ciencias de la computación, debido a las grandes repercusiones que habrían, en caso de encontrarse una solución. Si fuera cierta dicha pregunta, cualquier problema polinómicamente verificable sera polinómicamente decidible.

Es por ello, que el problema de definir la tratabilidad no es sencillo: no se sabe si $P = NP$ o $P \subseteq NP$. Esto es, puede ser que todos los problemas en NP tengan solución determinística polinomial; o puede ser que no sea equivalente el tiempo determinístico con el no determinístico.

La mayora de los investigadores creen que estas clases no son iguales, debido a que se han realizado bastantes esfuerzos para encontrar algoritmos de tiempo polinomial para varios problemas en NP, sin éxito. También han tratado de probar que las clases son distintas, pero eso conllevaría a demostrar que no existe un algoritmo "eficiente" para reemplazar la búsqueda por fuerza bruta.

Como señala el argumento filosófico del científico del MIT Scott Aaronson en su página web [1]: *"Si P=NP, entonces el mundo sería un lugar totalmente diferente al que solemos ver. La creatividad no tendría gran valor debido a que no habría ninguna diferencia entre resolver un problema y reconocer la solución una vez que ésta se encuentra. Cualquiera que pudiese apreciar una sinfonía podría ser Mozart; cualquiera que pudiese seguir un razonamiento paso a paso podría ser Gauss; cualquiera que pudiese reconocer una buena inversión podra ser Warren Buffett. Desde los términos darwinianos, si nosotros habitáramos en este tipo de universo, ¿por qué no habríamos tomado ya ventaja de ello?"*.

Clase NP-completo

Los problemas NP-completos (NPC) son aquellos que cumplen las siguientes condiciones:

- Son NP.
- Todos los demás problemas NP pueden ser "traducidos" a él con rendimiento polinomial.

Con la afirmación anterior, para averiguar si $P = NP$ es suficiente con tratar un problema en NPC. Si uno de los ejercicios que pertenecen a NPC tiene solución polinomial, entonces todos los que pertenecen a NP también la tienen, con lo que se cumpliría que $P = NP$. En otro caso, si se demuestra que no puede existir una solución polinomial para alguno de ellos, entonces ningún problema de los NPC la tiene; y se cumpliría que $P \subset NP$

Con todo esto, el destino de un problema NPC es el destino de todos: o bien todos son tratables o todos son intratables.

Clase NP-difícil

La clase de complejidad NP-*hard* (o NP-complejo, o NP-difícil) es el conjunto de los problemas de decisión que contiene los ejercicios H tales que todo problema L en NP puede ser transformado polinómicamente a H. Es parecido al concepto de NPC, pero en este caso, los problemas NP-difícil no tienen por qué estar en NP. Por tanto, un problema NPC es un problema NP-difícil y NP.

La clase NPC puede definirse alternativamente como la intersección entre NP y NP-*hard* (ver figura 2.1).

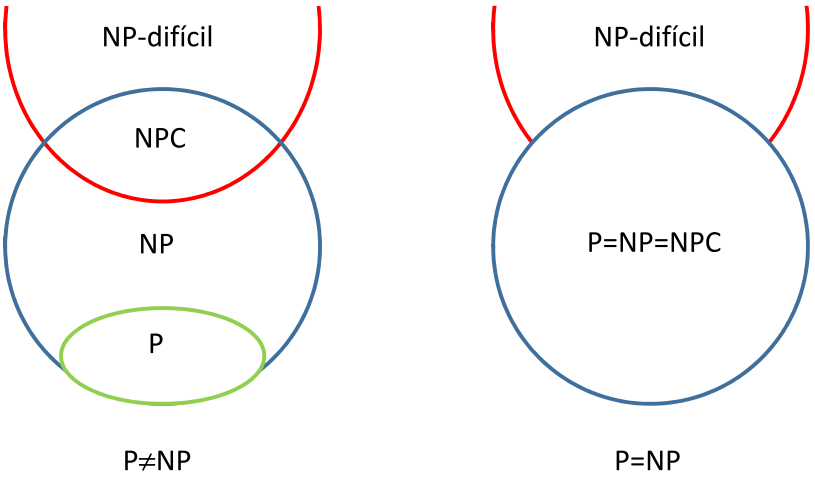

Figura 2.1: Diagrama de Euler para los problemas P, NP, NPC y NP-difícil.

2.3. Problemas de optimización

2.3.1. Modelos matemáticos

Como se comenta en [203], a pesar de que cualquier intento de clasificación tiene el inconveniente de ser esquemático y reduccionista, se podría adoptar la organización que sugiere Mark Meerschaert en su libro *Mathematical Modeling* [174]. Según él, la gran mayoría de los modelos matemáticos pertenecen a una de las siguientes categorías: modelos de optimización, dinámicos y probabilísticos.

Un modelo dinámico es aquel que depende del tiempo; en el probabilístico existe incertidumbre y, por último, el de optimización consiste en determinar el valor óptimo de un grupo de variables. La realidad es sumamente compleja, por lo que al tratar de modelarla, es posible que se requiera la combinación de varios tipos de modelos a la vez. Problemas que involucren la determinación del óptimo de una función aparecen muy frecuentemente cuando se configura un modelo matemático. No importa el tipo de problema en estudio, siempre se deseará maximizar los beneficios y minimizar los riesgos.

2.3.2. Problemas de optimización combinatoria

Desde el punto de vista matemático, un problema de optimización P se puede formular como la minimización (o maximización) de una función f sujeta a una serie de restricciones g_i, h_j, l_k.

$$P = \begin{cases} minf(x) & \text{función objetivo} \\ st & \text{retricciones} \\ g_i(x) \geq 0 & i = 1, ..., m \\ h_j(x) \geq 0 & j = 1, ..., p \\ l_k(x) \geq 0 & j = 1, ..., q \end{cases} \tag{2.2}$$

donde f, g_i, h_j, l_k son funciones generales del parámetro $x \in \mathbb{R}^n$. Nótese que se puede dar una definición alternativa para problemas de maximización modificando $minf(x)$ por $maxf(x)$.

Sea F el conjunto de soluciones que satisface todas las restricciones del problema. $\forall \sigma \in F$ es denominada solución factible. Una solución factible se denomina óptima, si el coste de la función objetivo de dicha solución es el menor, si el problema es de minimización; o el mayor, si es de maximización, de entre todas las soluciones factibles $\sigma \in F$.

Los problemas de optimización combinatoria COP *Combinatorial Optimization Problem* $COP = (S; f)$ se podría definir, tal y como se describe en [35], como la maximización o minimización de $f(x)$, con S representando al espacio de soluciones factibles del problema.

$$COP = \begin{cases} X = x_1, ..., x_n & \text{variables del problema} \\ D_1, ..., D_n & \text{dominios de las variables} \\ f : D_1 \times ... \times D_n \rightarrow \mathbb{R}^+ & \text{función objetivo} \\ S = \{s_i = \{(x_1, v_1), ..., (x_n, v_n)\}\} \mid v_i \in D_i & \text{espacio de soluciones} \end{cases} \tag{2.3}$$

donde cada variable x_i toma un valor v_i dentro del dominio D_i $(i = 1, ..., n)$, y cada solución $s_i \in S$ cumple con las restricciones del problema.

Óptimo global

Por tanto, el objetivo de estos tipos de problemas consiste en encontrar una solución $s^* \in S$ tal que el valor de la función objetivo sea un óptimo global. A continuación, se define el concepto de óptimo global s^* para un problema $P = (X, D, f, S)$.

Un óptimo global s^* en el espacio de soluciones S para un problema de minimización (o maximización) es aquel que cumple lo siguiente:

$$\forall s \in S, f(s^*) \leq (\text{ó} \geq) f(S) \tag{2.4}$$

Por lo que el óptimo global s^* es una solución, no necesariamente única, tal que cumple con todas las restricciones del problema; y no existe otra solución $s \in S$, cuyo valor de la función objetivo sea mejor.

Vecindario y óptimo local

Existen técnicas que aplicadas a un problema de optimización, de forma general sólo alcanzan óptimos locales. Antes de dar una definición de los mismos, es necesario conocer ciertos conceptos de la estructura de vecindario.

Una estructura de vecindario es una función $N : S \rightarrow 2^s$ la cual asigna a toda solución $s \in S$, un conjunto de vecinos $N(s) \subseteq S$. $N(s)$ también es conocido como el vecindario de s.

Una solución óptima local, $\overline{s}$ mínimo o máximo local, según la función objetivo del problema de maximización (o minimización) respecto a un vecindario N, es un resultado tal que cumple con la siguiente fórmula:

$$\forall s \in N(\overline{s}) : f(s) \geq (\text{ó} \leq) f(\overline{s}) \tag{2.5}$$

Análogamente, $\overline{s}$ es un óptimo local estricto si satisface la siguiente expresión:

$$\forall s \in N(\overline{s}) : f(s) > (\text{ó} <) f(\overline{s}) \tag{2.6}$$

2.3.3. Programación lineal

Introducción

A partir de la Segunda Guerra Mundial, se desarrolló una técnica denominada programación lineal, cuya finalidad era resolver cierto tipo de problemas de asignación de recursos entre distintas actividades.

Tal y como se comenta en [55], la idea fundamental de la optimización sin restricciones es la de encontrar el mayor o el menor valor de una función objetivo en un espacio de búsqueda. En el caso de la figura 2.2, serían los puntos a y b respectivamente.

En cambio, la programación lineal (LP) es el procedimiento más común de aplicar la optimización con restricciones. Este tipo de optimización debe obtener el mejor punto de la

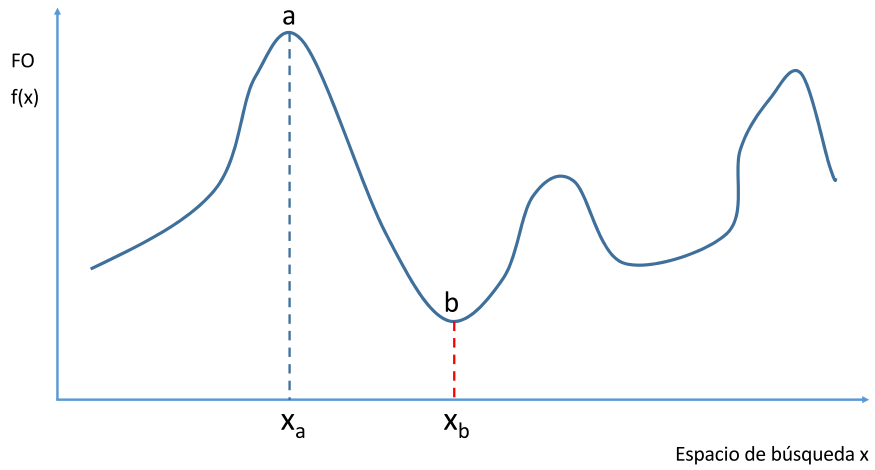

Figura 2.2: Optimización sin restricciones.

función. Pero a diferencia del caso anterior, se debe considerar diversas limitaciones en el proceso.

LP es un algoritmo matemático mediante el cual se resuelve un problema indeterminado, formulado a través de un sistema de inecuaciones lineales, optimizando una función objetivo, también lineal. Básicamente, consiste en optimizar (minimizar o maximizar) dicha función objetivo, de tal forma que las variables de la función se encuentren sujetas a una serie de restricciones, que se expresan mediante un sistema de inecuaciones lineales. El método tradicionalmente empleado para resolver problemas de programación lineal es el método simplex.

Elementos del problema

Los principales elementos de cualquier problema de optimización son los siguientes:

- Variables (o variables de decisión). Los valores de estos elementos son desconocidos al comienzo del proceso de optimización. Representan entidades que deben ser controladas o ajustadas.
- Función objetivo. Es una expresión matemática que combinan las variables para expresar una finalidad. Se puede requerir que se maximice o minimice dicha función.
- Restricciones. Son también expresiones matemáticas que incluyen y combinan las variables para reflejar limitaciones a las posibles soluciones del problema.
- Límites en variables. Únicamente en casos excepcionales, se permiten que las variables en los problemas de optimización tomen cualquier valor (por ejemplo, de $-\infty$ a $+\infty$). En otro caso, se deben definir límites a las variables del problema.

En LP, todas las expresiones matemáticas relativas a la función objetivo, así como las restricciones son lineales. El término "programación" se debe a un uso arcaico de la palabra que

debería significar "planificación". Por esta razón, la programación lineal se debera considerar como una planificación con modelos lineales.

Formulación algebraica general

En [201] se desarrolla un problema de programación lineal de la siguiente manera: Determinar $x = (x_1, ..., x_n) \in \mathbb{R}^n$ tal que

$$\text{máx (ó mín)} \quad Z = \sum c_1 x_1 + c_2 x_2 + ... + c_n x_n \tag{2.7}$$

sujeto a

$$a_{11}x_1 + a_{12}x_2 + ... + a_{1n}x_n \quad (\leq, \geq, \text{ó} =) \quad b_1 \tag{2.8}$$
$$a_{21}x_1 + a_{22}x_2 + ... + a_{2n}x_n \quad (\leq, \geq, \text{ó} =) \quad b_2 \tag{2.9}$$
$$...$$
$$a_{m1}x_1 + a_{m2}x_2 + ... + a_{mn}x_n \quad (\leq, \geq, \text{ó} =) \quad b_m \tag{2.10}$$

Si en el modelo anterior, todos los signos son igualdades, y los elementos $b_i \geq 0$, se dice que el problema se encuentra en formato "estándar".

Como resumen, la formulación estándar tiene las siguientes características:

- La función objetivo debe ser maximizada o minimizada.
- Todas las restricciones son de igualdad.
- Todas las restricciones tienen la parte derecha de la expresión matemática valores no negativos.
- Todas las variables se restringen a tomar valores no negativos.

Solución gráfica del problema lineal e interpretación

Este procedimiento es válido para resolver problemas de dos o tres variables. Las fases de este proceso son las siguientes:

1. Se crea un sistema de coordenadas cartesianas en el que las variables de decisión están determinadas por los ejes.
2. Se representa en el sistema coordenado las restricciones del problema. Cada restricción define una región. La intersección de todas las regiones obtenidas determina la región factible o espacio de soluciones. Este es un conjunto convexo; es decir, que para cualquier par de puntos del conjunto, el segmento que los une también se encuentra contenido en él.
3. Se determinan los puntos extremos. Estos se encuentran en el borde de la región factible, y son los candidatos para ser evaluados como la solución óptima.

En el ejemplo de la figura 2.3, aparece la región factible de un problema lineal para dos variables de decisión x e y.

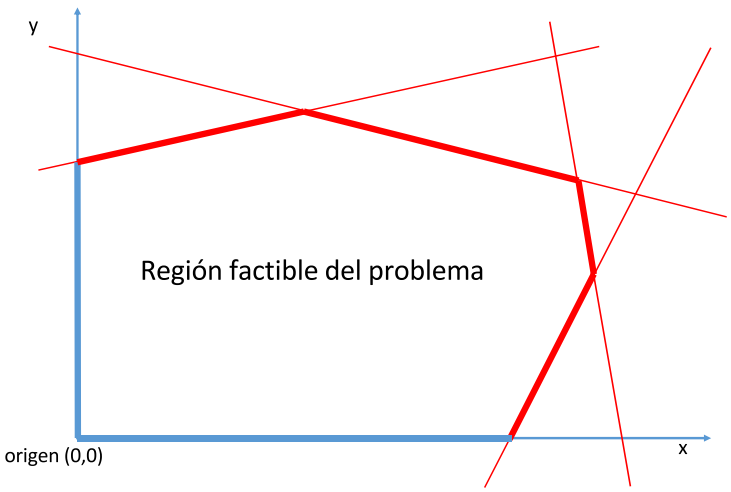

Figura 2.3: Región factible de un problema lineal.

2.3.4. Programación entera

Como se ha comentado en el punto anterior, las variables de decisión de un modelo de programación lineal toman valores reales, de tal forma que estos modelos se consideran contínuos. Pero existen casos en los que el problema demanda variables enteras en el modelo correspondiente. De esta manera, se puede considerar el siguiente problema de optimización:

$$\text{máx } Z = \sum_{j=1}^{n} c_j x_j \tag{2.11}$$

sujeto a

$$\sum_{j=1}^{n} a_{ij} x_j = b_i \quad (i = 1, 2..., m) \tag{2.12}$$

$$x_j \geq 0 \quad (j = 1, 2..., n) \tag{2.13}$$

$$x_j \quad (j = 1, 2..., n) \tag{2.14}$$

$$x_j \in \mathbb{Z} \quad (\text{para algunos o todos } j = 1, 2..., n) \tag{2.15}$$

Este modelo se denomina problema de programacion entera (*Integer Linear Programming Problem -ILP-*). Se nombra como programación entera-mixta (*Mixed Integer Program*) cuando alguna, pero no todas sus variables, se restringen a tomar valores enteros. En el caso de que todas las variables sean de tipo entero, el modelo se llamaría programación entera pura (*Pure Integer Program*).

En los casos en los que se tenga que realizar una decisión del tipo "sí" / "no", se utilizarán variables cuyo valores se restringe a 0 y 1. A este tipo de variables se las conocen como binarias, lógicas o simplemente variables 0-1.

En el ejemplo de la figura 2.4 aparece la región factible de un problema entero puro. Como se puede apreciar, únicamente los puntos rojos son soluciones factibles del problema.

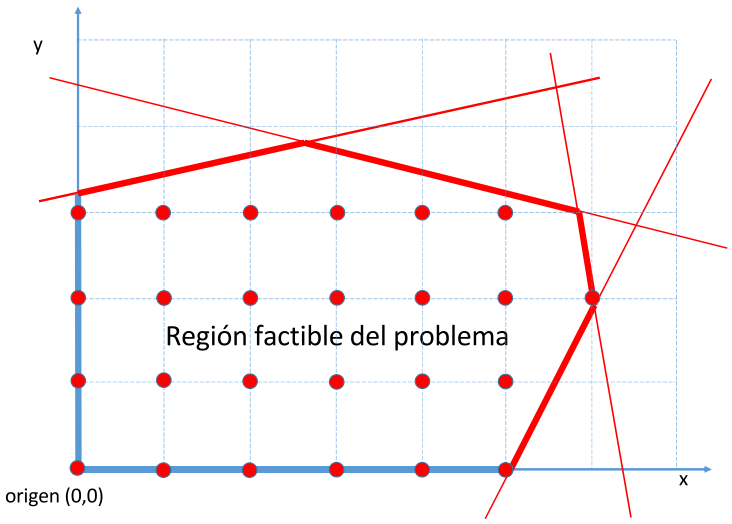

Figura 2.4: Región factible con una programación entera.

Relajación lineal

Se considera la programación entera como un subconjunto de la lineal, en el sentido de que las variables, algunas o todas, del modelo deben ser de tipo entero. Por otra parte, un modelo entero en el que se elimina las restricciones de variables enteras, se transforma en un programa lineal asociado. Este proceso se denomina relajación lineal (*Linear Relaxation* -LR-). A continuación, se enumeran algunas características básicas en relación al programa entero (ILP), y a su correspondiente relajación lineal (LR).

- En problemas de minimización: El valor objetivo óptimo del LR es menor o igual al correspondiente del ILP.
- En problemas de maximización: El valor objetivo óptimo del LR es mayor o igual al correspondiente del ILP.
- Si LR es infactible, entonces ILP también lo es.
- Si LP obtiene una optimización mediante valores enteros, entonces la solución es también factible y óptima para ILP.

Capítulo 3

Métodos de resolución

3.1. Introducción

La resolución de los problemas de optimización combinatoria podría llevarse a cabo de la siguiente forma: dada una instancia del problema, se deberían buscar todos los resultados en el espacio de soluciones, para finalizar eligiendo la mejor. A pesar de que esta manera de resolver el ejercicio parece simple y lógica, en la mayoría de los casos la propuesta se vuelve rápidamente impracticable, debido a que el número de posibles soluciones crece de manera exponencial según el tamaño de la instancia. En algunos problemas de optimización, el estudio y análisis de la estructura del problema, así como las características de las instancias correspondientes, permiten desarrollar algoritmos que alcanzan soluciones óptimas en menor tiempo computacional que una búsqueda exhaustiva o completa. En otros problemas, en cambio, estos mismos procedimientos no logran un rendimiento mucho mejor que la búsqueda exhaustiva.

Dada la importancia práctica de estos tipos de problemas, se han desarrollado diferentes algoritmos con el objeto de obtener soluciones a los mismos. Éstos se clasifican en exactos y metaheurísticos. Los métodos exactos garantizan obtener la solución óptima a un problema de optimización en un tiempo computacional que dependerá de la instancia del problema, siendo la misma de tamaño finito. Sin embargo, para muchos problemas complejos (NP, NP-completo, NP-difícil), los algoritmos exactos necesitan un tiempo exponencial para alcanzar las soluciones óptimas.

Incluso para instancias pequeñas, estos procedimientos podrían consumir demasiado tiempo de ejecución, lo cual resultaría poco práctico.

Por otra parte, los métodos metaheurísticos obtienen soluciones cercanas a la solución óptima, pero con menor coste computacional.

Por tanto, los algoritmos exactos son procedimientos que obtienen la solución óptima de un problema de optimización. Dependiendo del tipo de problema, podría darse el caso que para entradas al proceso de grandes dimensiones, el tiempo de ejecución tendería a ser inviable (en caso de problemas NP). En los siguientes apartados del capítulo, se presentan algunos métodos de optimización siguiendo las siguientes razones:

- El procedimiento forma parte de alguna matheurística mencionada en la sección 3.4 de

este capítulo , como la ramificación y poda;

- Porque son importantes para entender alguna otra técnica de resolución exacta: por ejemplo, el método simplex se suele emplear para aplicar el método de ramificación y corte.

- O bien porque será utilizado para la resolución del problema EDP: la matheurística desarrollada en este documento se modela con un componente ILP, implementado para ser resuelto mediante la herramienta CPLEX.

3.2. Métodos Exactos

3.2.1. Método simplex

Introducción

El algoritmo o método simplex es un procedimiento para resolver problemas de programación lineal. Para llevar a cabo este proceso, se va creando una serie de restricciones lineales, obteniéndose un poliedro como una región factible (si el problema se encuentra definido únicamente por dos variables x e y, la región sería un polígono conexo -ver figura 2.3-). El algoritmo comienza en un vértice y se mueve a lo largo de las aristas del poliedro hasta que alcanza el vértice con la solución óptima.

Cuando todas las variables son estrictamente positivas, la solución resultante debe mejorar la función objetivo, pudiendo terminar el algoritmo después de un número finito de pasos. Esta afirmación se sustenta debido a que el poliedro tiene un número determinado de vértices. En cambio, a los resultados donde alguna de las variables toma el valor de cero, se les denomina soluciones "degeneradas". Éstas incluyen una alta posibilidad de que no mejoren la función objetivo.

Otra característica importante, pero menos probable, de las soluciones degeneradas, es la posibilidad de que se obtengan los mismos valores asignados a las variables de la solución más de una vez, produciéndose, por tanto, un bucle infinito. Para evitar este caso, se suelen añadir nuevas reglas.

En general, este algoritmo es eficiente, y puede garantizar la obtención del óptimo global si se siguen ciertas precauciones, en torno fundamentalmente, a evitar los ciclos en el proceso. Sin embargo, tiene un pobre rendimiento en los peores casos del problema. Asímismo, no puede asegurar que se resuelva un problema en tiempo polinómico.

Método de las variables artificiales

Como se verá posteriormente, en la fase inicial del método simplex se necesita disponer de una solución factible. Uno de los procedimientos más utilizados para llevar a cabo este cometido es el método de las variables artificiales [201].

Variables de holgura y artificiales Pero antes de aplicar este proceso, sería necesario eliminar todas las desigualdades de las restricciones de la formulación, para obtener una forma estándar. Para ello, se utilizan las variables de holgura s_i y t_i.

$$\sum_{j=1}^{n} a_{ij}x_j \leq b_i \qquad \text{se puede poner} \qquad \sum_{j=1}^{n} a_{ij}x_j + s_i = b_i \tag{3.1}$$

$$\sum_{j=1}^{n} a_{ij}x_j \geq b_i \qquad \text{se transforma en} \qquad \sum_{j=1}^{n} a_{ij}x_j - t_i = b_i \tag{3.2}$$

De esta manera, se puede considerar s_i como la cantidad del recurso b_i que no se ha utilizado. Por otra parte, t_i se interpreta como el exceso del recurso que se emplea respecto a la cantidad b_i.

Volviendo al método de las variables artificiales, se debe comenzar obteniendo la forma estándar de las restricciones del problema lineal (añadiendo variables de holgura). A continuación, se suman variables artificiales $r_i \geq 0$ a las restricciones que originariamente fueron igualdades y a aquellas que tenían el signo $\geq$, y por tanto, se les incorporó variables de holgura con signo negativo.

$$\sum_{j=1}^{n} a_{pj}x_j \geq b_p \qquad \text{pasará a ser} \qquad \sum_{j=1}^{n} a_{pj}x_j - t_i + r_p = b_q \tag{3.3}$$

con t_p variable de holgura y r_p artificial. Si la restricción es

$$\sum_{j=1}^{n} a_{qj}x_j = b_q \qquad \text{se sustituye por} \qquad \sum_{j=1}^{n} a_{qj}x_j + r_p = b_q \tag{3.4}$$

Se debe notar que así como las variables de holgura tienen una interpretación económica, las variables artificiales se agregan únicamente para poder iniciar de manera sencilla el método del simplex, por lo que se introducen por conveniencia matenática.

Base y soluciones básicas

Para poder tratar con la optimalidad y la mejora de la solución básica factible en la fase II del algoritmo, es necesario tener en cuenta algunos conceptos, como puede ser la base de un programa lineal.

Teniendo un sistema de ecuaciones simultáneas:

$$a_{11}x_1 + a_{12}x_2 + ... + a_{1n}x_n = b_1 \tag{3.5}$$

$$a_{21}x_1 + a_{22}x_2 + ... + a_{2n}x_n = b_2 \tag{3.6}$$

$$...$$

$$a_{m1}x_1 + a_{m2}x_2 + ... + a_{mn}x_n = b_m \tag{3.7}$$

se puede escribir en forma matricial:

$$Ax = b \tag{3.8}$$

donde

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix} \tag{3.9}$$

$$x = (x_1, x_2, ..., x_n)^T \tag{3.10}$$

$$b = (b_1, b_2, ..., b_m)^T \tag{3.11}$$

Si se supone que a_i es el vector columna i-ésimo de A y $r(A) = m$ [1]; y se toma cualquier submatriz B de A no singular [2] de orden m, igualándose a cero las restantes $n - m$ variables asociadas a los vectores columna de A que no se encuentran en B, se tiene el siguiente sistema resultante:

$$Bx_B = b \tag{3.12}$$

de m ecuaciones con m variables, obteniéndose la solución "básica" x_B. Si alguna variable básica, contenida en el sistema toma el valor cero, la solución se denomina "degenerada".

Por último, cualquier vector columna de $a_i \in A$ se puede poner como combinación lineal de los elementos de la base B:

$$y_i = B^{-1}a_i \tag{3.13}$$

donde y_i es la relación de los diferentes columnas de la base para obtener de forma lineal a_i.

Algoritmo

Sin entrar en detalle, el método obtiene la solución fundamentalmente en dos etapas:

- Fase I. En el primero de ellos, se busca un punto solución inicial. Dependiendo de la naturaleza del programa o modelo, puede ser un proceso trivial o debe ser necesario aplicar el algoritmo simplex a una versión modificada del programa original (por ejemplo, añadiendo variables de holgura mediante una matriz identidad al modelo inicial). Los posibles resultados de esta fase pueden ser, por una lado una solución viable; y por otro una región factible vacía. En este último caso, se considera al programa lineal infactible.

  De cualquier forma, la búsqueda de la solución óptima se realizará examinando únicamente los puntos extremos del espacio de soluciones.

- Fase II. En este caso, se aplica el algoritmo simplex empleando la solución factible obtenida en la fase I como el punto inicial.

[1] rango de la matriz: número de filas o columnas linealmente independientes
[2] matriz singular: matriz cuadrada cuyo determinante es igual a cero

- Para ello, se debe determinar si el paso a una solución básica factible adyacente puede mejorar el valor del objetivo. Si es así, se debe ir al paso siguiente. En otro caso, se ha alcanzado la solución óptima.
- Y finalmente se debe resolver la solución básica factible adyacente con mayor mejora en el valor del objetivo. Volver al paso anterior (de la fase II) y repetir el proceso hasta alcanzar una solución óptima; o bien llegar a un problema infactible o no acotado.

Optimalidad y mejora de la solución básica factible

Dada una base B no óptima, se debe conocer las reglas para, por un lado, escoger la variable (o vector columna asociado) que deja la base; por otra parte, para determinar cuál es la variable de entrada a B; y por último, para averiguar si es posible una mejora del objetivo [201].

Regla de la variable de salida (que abandona la base) . Dada la solución básica factible $x_B = B^{-1}b$, si el vector columna a_j de fuera de la base tiene $y_{ij} > 0$ para algún i, entonces puede entrar en la base en lugar de un vector b_k de la base que minimice el ratio $\frac{x_{Bk}}{ykj}$ tal que $\frac{x_{Bk}}{ykj}, y_{ij} > 0$

Regla de la variable que entra en la base y regla de parada . Tanto la regla de selección de la variable de entrada, como el criterio de optimalidad para la solución obtenida, se basan en el valor denominado "coste reducido" (o "coste relativo"). $z_j - c_j$ es el coste reducido de la variable x_j, que indica el cambio del valor del objetivo si la variable no básica x_j se incrementa de 0 a 1.

Por tanto, dada la solución básica factible $x_B = B^{-1}b$, con un valor para la función objetivo $z = c_B^T x_B$, la actual solución es óptima si $z_j - c_j \geq 0$ para toda columna a_j de A.

Por último, la variable de entrada en la base será aquella que contenga el $z_j - c_j$ más negativo.

3.2.2. Método del punto interior

El método del punto interior (*Interior Point Method* -IPM-), es una clase de algoritmos que resuelven problemas de optimización convexos lineales y no lineales.

John Von Neumann [71] planteó un IPM aplicado a la programación lineal, pero no era eficiente; no resolvía en tiempo polinómico. De hecho, era más lento que el simplex. Va a ser Narendra Karmarka [128] el que cree un método con su nombre para resolver problemas lineales de una forma más eficiente (en tiempo polinómico).

A diferencia del algoritmo simplex, que sigue una secuencia de puntos extremos y adyacentes hasta llegar al resultado óptimo, pero pudiendo crecer el número de iteraciones de forma exponencial en problemas de grandes dimensiones; IPM se mueve desde el interior de la región factible hasta la solución óptima en tiempo polinómico.

3.2.3. Algoritmo por retroceso

Vuelta atrás, retroceso o *Backtracking* es una estrategia para encontrar soluciones a problemas que satisfacen restricciones. El término "*backtrack*" fue acuñado por primera vez por el matemático estadounidense D. H. Lehmer en la década de 1950.

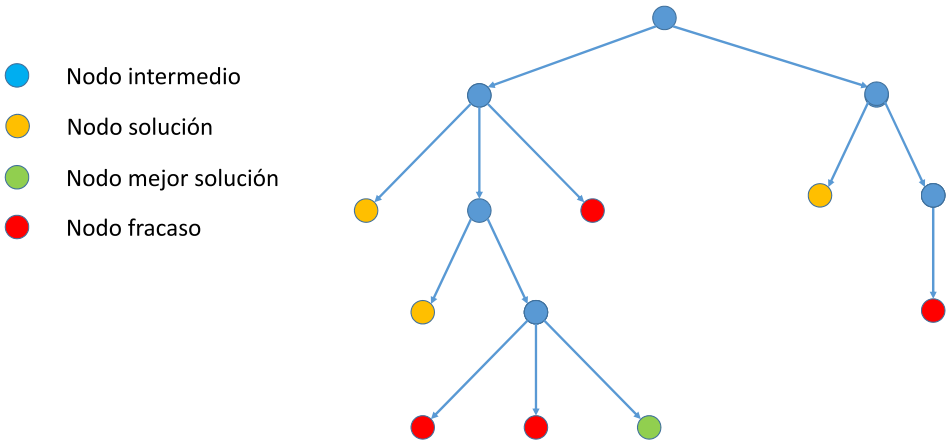

Figura 3.1: Árbol de expansión con Backtracking.

En el algoritmo 3.1 aparece la función recursiva que resuelve el problema para una única solución. Es decir, que si encuentra una solución, finaliza.

Algoritmo 3.1: `Backtracking`

```
Input : step
Output : solution
 1: initialize()
 2: repeat
 3:    option ← selectNewOption()
 4:    if accept(option) then
 5:       solution′ ← add(solution, option)
 6:    end if
 7:    if not isComplete(solution) then
 8:       nextStep ← searchSpace(step)
 9:       success ← Backtracking(nextStep)
10:    end if
11:    if not success then
12:       solution′ ← deleteOptions(solution, option)
13:       {in other case → is complete solution}
14:    end if
15: until success or isLast(option)
16: return solution
```

Como se comenta en [110], dentro de las técnicas de diseño de algoritmos, el método de vuelta atrás es uno de los de más amplia utilización, en el sentido de que se puede aplicar a la resolución de un gran número de problemas, muy especialmente en aquellos de optimización.

En su forma básica, la idea de *Backtracking* se asemeja a un recorrido en profundidad dentro de un grafo. El grafo en cuestión suele ser un árbol, o por lo menos no contiene ciclos.

Vuelta atrás proporciona una manera sistemática de generar todas las posibles soluciones siempre que dichos resultados sean susceptibles de resolverse en etapas.

Tal y como se ha indicado en líneas anteriores, la vuelta atrás se asemeja a un recorrido en profundidad dentro de un árbol, cuya existencia sólo es implícita, y que denominaremos árbol de expansión. Esta estructura es conceptual y únicamente se hará uso de su organización como tal, en donde cada nodo de nivel k representa una parte de la solución y está formado por k etapas que se suponen ya realizadas.

Sus hijos son las prolongaciones posibles al añadir un nuevo paso. Para examinar el conjunto de posibles soluciones, es suficiente con recorrer este árbol, construyendo soluciones parciales a medida que se avanza en el recorrido. Es decir, el resultado es expresable en forma de secuencia de decisiones. Además, existe una función que permite averiguar si para dicha secuencia de decisiones, que es la solución actual, se cumple o no con las restricciones.

En este recorrido pueden suceder dos cosas: la primera de ellas es que tenga éxito si, procediendo de esta manera, se llega a una solución (una hoja del árbol, o nodo final). Si lo único que se buscaba era una solución al problema, el algoritmo finaliza aquí. Ahora bien, si el objetivo es encontrar todas las soluciones o la mejor de entre todas ellas, el algoritmo seguirá explorando el árbol en búsqueda de soluciones alternativas.

Es necesario disponer de otra función que permita determinar si una secuencia de decisiones es solución al problema planteado. Por otra parte, el recorrido no tiene éxito si en alguna etapa la solución parcial construida hasta el momento no se puede completar; entonces nos encontramos en lo que se denomina un "nodo fracaso". En tal caso, el algoritmo vuelve hacia atrás en su recorrido, eliminando los elementos que se hubieran añadido en cada etapa, a partir de ese nodo. En este retroceso, si existe uno o más caminos aún no explorados que puedan conducir a una solución, el recorrido del árbol continúa por ellos. En la figura 3.1 se puede apreciar el ejemplo de un árbol de búsqueda para este algoritmo. En esta ocasión, se busca la mejor solución, representado por el nodo verde.

3.2.4. Ramificación y acotación

El método de diseño de algoritmos de ramificación y acotación (también llamado ramificación y poda) es una variante del procedimiento de *Backtracking* mejorado sustancialmente. El término (del inglés *Branch & Bound*) se aplica mayoritariamente para resolver cuestiones o problemas de optimización [110].

Esta técnica se suele interpretar como un árbol de soluciones, donde la exploración de ésta nos lleva a la obtención de posibles soluciones. La característica principal de este procedimiento con respecto al anterior, se basa en que este algoritmo se encarga de detectar en qué ramificación las soluciones dadas ya no son óptimas para, de esta forma, podar esa rama y no continuar malgastando recursos y procesos.

Ramificación y acotación, al igual que en el proceso de vuelta atrás, realiza una enumeración parcial del espacio de soluciones, basándose en la generación de un árbol de expansión.

Asímismo, esta técnica utiliza cotas para podar aquellas ramas del árbol que no conducen a la solución óptima. Para ello, se calcula en cada nodo una cota con el posible valor de aquellas soluciones alcanzables desde él. Si la cota muestra que cualquiera de estas soluciones tiende

a ser necesariamente peor que el mejor resultado hallado hasta este momento, no se necesita seguir explorando por esa rama del árbol, lo que permite aplicar el proceso de poda.

Para cada nodo del árbol se dispondrá de una función de coste que estime el valor óptimo de la solución si se continúa por ese camino. De tal manera que si la cota que se obtiene para un nodo es peor que un resultado previamente obtenido por otra rama, entonces se puede acotar la rama por el nodo en cuestión; debido a que no es interesante seguir por él: no se obtiene un mejor resultado. Evidentemente no se puede aplicar la poda sobre ningún elemento hasta que no se haya encontrado alguna solución. Las funciones de coste son crecientes respecto a la profundidad del árbol. Es decir, si h es una función de coste, entonces $h(n) \leq h(n'), \quad \forall n'$ nodo descendiente de n.

Básicamente, este procedimiento se realiza en varias etapas:

1. La primera de ellas, denominada de selección, se encarga de tomar un nodo de entre los que se encuentren vivos. La forma de escogerlo va a depender directamente de la estrategia de búsqueda que se decida para el algoritmo.

2. En la segunda etapa, la ramificación, se construyen los posibles nodos hijos del nodo seleccionado en el paso anterior.

3. En el tercer paso, la poda, se eliminan algunos de los nodos creados en la etapa anterior. Esto contribuye a disminuir en lo posible el espacio de búsqueda, y así atenuar la complejidad de estos algoritmos basados en la exploración de un árbol de posibilidades.

4. A continuación, en la siguiente fase, aquellos nodos no podados pasan a formar parte del conjunto de nodos vivos, y se comienza de nuevo por el proceso de selección.

5. El algoritmo finaliza cuando encuentra la solución, o bien cuando se agota el conjunto de nodos vivos.

Variante mediante relajación lineal de un problema entero

Este método tiene gran importancia en la resolución de problemas enteros (ILP). Una posible estrategia podría ser la obtención, relajando el problema, de una solución lineal. Seguídamente, se deberían escoger las variables con resultados no enteros y "convertirlas" en valores enteros. Con algo más de detalle, se podrían seguir las siguientes etapas para un problema de maximización [201]:

- Paso 0. Iniciación. Se resuelve el problema lineal relajado (LR) asociado al problema entero. Si la solución óptima es entera, se para, debido a que es también la solución del problema entero. En otro caso, se debe determinar una cota inferior z_i para el valor óptimo del objetivo de un problema de maximización. Esta variable podría tomar bien el valor -∞, o bien el valor de la función objetivo en algún punto factible.

- Paso 1. Ramificación. Mediante alguna estrategia de ramificación, se selecciona un subconjunto, o subproblema, de soluciones factibles que quede sin sondear; y una componente no entera de la solución del subproblema en cuestión. Se haría una partición

en el subconjunto elegido obteniendo otros dos más pequeños, obtenidos al añadir restricciones que excluyan los valores fraccionarios (no enteros) de la componente elegida.

- Paso 2. Acotación. Para cada nuevo subconjunto, se tiene que determinar una cota superior z_s para el valor del objetivo del programa entero.
- Paso 3. Sondeo. Analizar los subconjuntos que puedan contener la solución óptima y considerar como terminales aquellos que cumplan alguna de las siguientes condiciones:
  - El subconjunto es infactible, o
  - $z_s \leq z_l$ o
  - z_s se alcanza en un punto factible para el programa entero y $z_s > z_l$

  En el último caso, se debe establecer $z_l = z_s$ e ir al paso 3 para buscar en otros subconjuntos.
- Paso 4. Convergencia. Si todos los subconjuntos son terminales, parar y tomar como solución z_l. En otro caso, ir al paso 1.

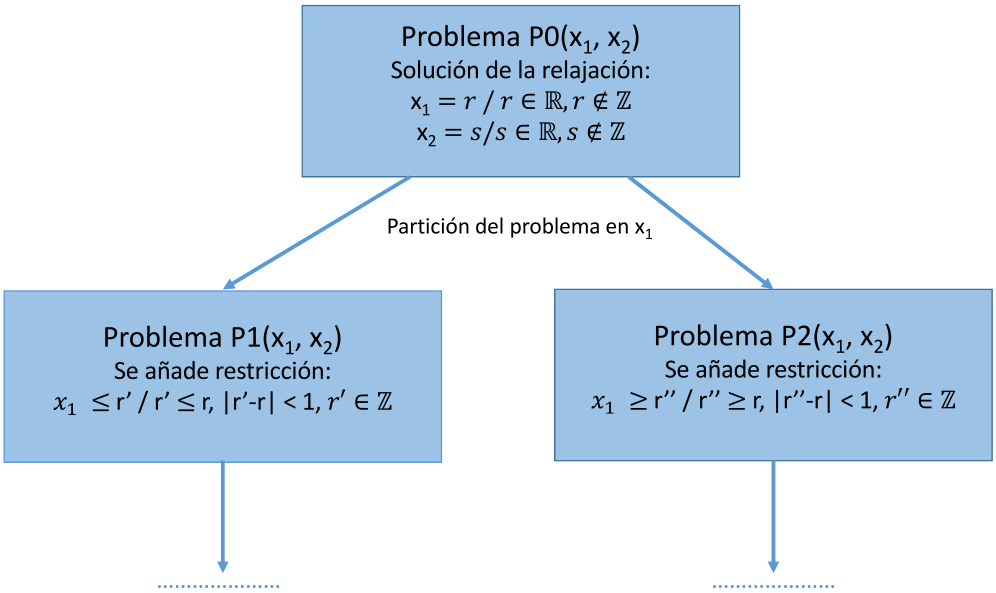

Figura 3.2: Progreso de un algoritmo de ramificación y acotación con relajación inicial.

En la figura 3.2 aparece la evolución del algoritmo *Branch & Bound*, que resuelve un problema entero, y en la que se aplica una relajación lineal inicial. Primeramente, cuando se tiene un problema $P0$, que consta de dos variables x_1 y x_2, se trata de resolver su relajación lineal. En este caso, ambas variables obtienen valores con decimales. Para seguir con la búsqueda, en el ejemplo se escoge el primero de ellos (x_1) y se divide el problema en dos ($P1$ y $P2$); añadiendo en cada caso una restricción adicional. En el primer problema ($P1$), x_1 debe ser menor o igual que el valor entero inmediatamente menor al conseguido por esta misma variable, mediante la relajación de P0. En $P2$ en cambio, x_1 debe poseer un valor mayor o igual al entero inmediatamente superior al valor real obtenido en $P0$ con la relajación.

3.2.5. Método de planos de corte

En optimización matemática, el método *Cutting Plane* refina un conjunto factible o función objetivo mediante inecuaciones lineales. Se suele utilizar para encontrar soluciones enteras en problemas enteros mixtos (MILP). Fue introducido por Ralph E. Gomory.

La idea de este enfoque consiste en ir resolviendo una sucesión de problemas de programación lineal relajados con una región factible cada vez más restringida, hasta alcanzar una solución óptima con valores enteros [201]. Si la solución x^* óptima del programa lineal relajado no tiene todos sus componentes enteros, es posible obtener una desigualdad (corte o restricción) válida para la región factible del problema entero que no incluya a la solución x^*. Dicha desigualdad se incorporaría al programa lineal, obteniendo, por tanto, una región factible más reducida. Estre proceso se repetiría hasta alcanzar, si existe, una solución con valores enteros.

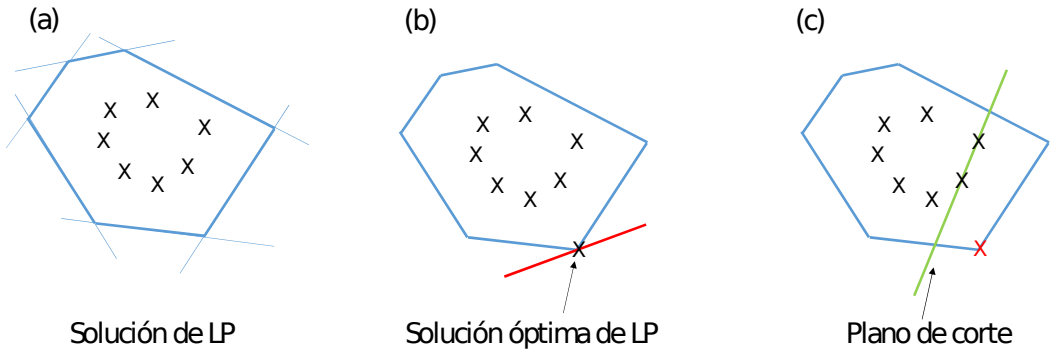

Figura 3.3: Aplicación del método de planos de corte a un modelo de dos variables.

Este procedimiento trabaja con problemas MILP, resolviendo el correspondiente programa lineal no entero; es decir la relajación lineal del programa entero.

Algoritmo 3.2: `Cutting Plane`

```
Input : P {an integer program}
Output : x*
 1: LR ← calculateLR(P) {Linear relaxation LR from P and solve it}
 2: repeat
 3:    x* ← solveLR(LR)
 4:    if isInteger(x*) then
 5:       return x*
 6:    end if
 7:    newEquation ← findEquation(x*) {Find equation which excludes x*}
 8:    LR ← addEquation(equation) {Add equation to LR }
 9: until endConditions()
```

La teoría de la programación lineal indica, que bajo ciertas asumpciones, se puede encontrar con toda seguridad un punto extremo óptimo. Se debe comprobar que el óptimo obtenido es una solución entera. En caso de que no sea así, se asegura la existencia de una inecuación lineal que separa el óptimo de la envolvente convexa del conjunto factible. Se puede añadir dicha inecuación (corte) al programa lineal relajado. Entonces, la solución no entera actual ya no es

factible para la relajación. Este proceso se repite hasta que se encuentra una solución entera óptima. En el algoritmo 3.2 se encuentra un breve resumen del pseudocódigo.

En la figura 3.3 aparece el resultado de aplicar este proceso a un modelo de dos variables. En (a) se puede apreciar el polígono que contiene la solución del LP relajado (donde las "equis" son soluciones internas). En (b) se calcula la solución óptima. Al no ser una solución entera, hay que aplicar el método del punto de planos de corte. En (c) se genera un corte (inecuación lineal) sobre la solución de la LR. Con la nueva restricción se vuelve a calcular una solución sobre el modelo relajado.

3.2.6. Ramificación y corte

A comienzo de los 80, se propuso un algoritmo para resolver programas enteros de grandes dimensiones [69] y tuvo bastante éxito. Este constaba de aplicar un algoritmo de planos de corte a la relajación lineal del problema, y luego resolver la relajación más ajustada con un algoritmo *Branch and Bound*. Esta técnica es ahora conocida como *Cut and Branch*. Ahora bien, si en cada nodo del árbol ejecutamos un algoritmo de planos de corte sobre la relajación lineal de dicho nodo, se pueden obtener cotas superiores más ajustadas, para un problema de maximización (o inferiores para el correspondiente ejercicio de minimización). Esta idea da origen al algoritmo *Branch and Cut*. Por tanto, como resumen, éste es un método exacto que consiste en una combinación del procedimiento de ramificación y poda y el método de planos de corte.

El método resuelve la programación lineal sin restricciones de enteros utilizando, por ejemplo, el algoritmo simplex. Cuando se obtiene una solución óptima, y ésta asigna valores no enteros a una variable que se supone que es entera, se puede utilizar un algoritmo *Cutting Plane*, proporcionando éste unas inecuaciones, que se añadirían al programa lineal. La nueva programación proporcionaría una solución menos "fraccional".

En la figura 3.4 aparece la evolución del algoritmo *Branch & Cut*. Gran parte del proceso se corresponde con la variante del algoritmo *Branch & Bound* que resuelve problemas enteros mediante una relajación inicial del problema (figura 3.2 de la sección 3.2.4). Únicamente habría que añadir a la explicación anterior que al finalizar $P2$, se ejecuta el método de planos de corte, incluyendo la restricción resultado en un nuevo problema $P3$.

En el algoritmo 3.3 aparece la función que implementa este algoritmo. Básicamente, se va tomando un problema P de una lista de problemas L. Se resuelve el problema relajado correspondiente, obteniéndose *sol*. Si la solución obtenida es factible y es mejor que la mejor solución actual y es entera, se actualiza la soluciones v^* (función objetivo de la mejor solución) y x^* (mejor solución) con los nuevos valores. Y se sigue probando con nuevos problemas de la lista L. En otro caso, se puede probar el algoritmo de planos de corte, para encontrar nuevas restricciones que se puedan incluir en el problema en curso P y así poder ajustar mejor la solución relajada. Si no se obtienen nuevas restricciones del procedimiento *Cutting Plane*, se puede dividir el problema en $P1, P2, ..., Pk$ y seguir con el bucle. Se devolvería la mejor solución x^*.

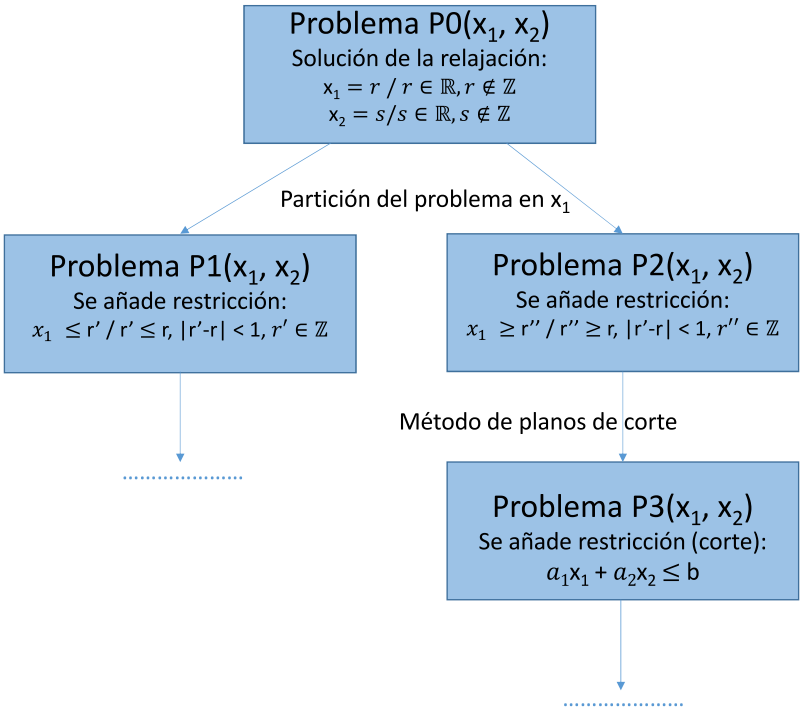

Figura 3.4: Progreso de un algoritmo de ramificación y corte.

Algoritmo 3.3: `Branch & Cut`

```
Input : ILP {initial integer program}
Output : (x*, v*)
 1: L ← add(L, ILP) {Initialize variables: insert initial ILP into L = active problem list}
 2: while not empty(L) do
 3:   P ← selectProblem(L)
 4:   repeat
 5:     sol ← solveLR(P)
 6:     if feasible(sol) then
 7:       (x, v) ← sol {v = objective value, x = calculated solution}
 8:       if v* <= v then
 9:         if isInteger(x) then
10:           v* ← v
11:           x* ← x
12:         else
13:           newEquations ← solveCuttingPlanes(x)
14:           P ← add(P, newEquations)
15:         end if
16:       end if
17:     end if
18:   until isEmpty(newEquations)
19:   L ← {P1, P2, ..., Pk} {Split the problem P, dividing into subproblems P1, P2, ..., Pk}
20: end while
21: return (x*, v*)
```

3.2.7. *Branch & Price*

Introducción

Branch and Price es un método de optimización combinatoria que resuelve problemas lineales enteros (ILP) y lineales enteros mixtos (MILP), aunque contengan una gran cantidad de variables. Básicamente es un híbrido del procedimiento de ramificación y acotación y el método de generación de columnas (*Column Generator Method*).

Este procedimiento tiene en cuenta algunos conceptos, como puede ser la formulación de un problema dual o el método mencionado anteriormente de generación de columnas.

Dualidad en la programación lineal

La idea fundamental de la dualidad en la programación lineal se basa en que cada programa lleva asociado otro "dual", de forma que la resolución del programa lineal original permite obtener una solución para su dual [201].

Teniendo en cuenta que un problema de programación lineal en forma simétrica de maximización cumple los siguientes puntos:

- El objetivo es de la forma de maximización.
- Todas las restricciones son desigualdades del tipo $\leq$.
- Las variables son no negativas.

Y para el problema de minimización en su forma simétrica, el objetivo cambia a minimización, y todas las desigualdades son de tipo $\geq$.

Se establece una relación "primal-dual" en forma simétrica como sigue. Sean los problemas de programación lineal:

$$\text{máx}\, z = c^T x \tag{3.14}$$

sujeto a

$$Ax \leq b \tag{3.15}$$

$$x \geq 0 \tag{3.16}$$

donde c es un vector $n \times 1$, b es un vector $m \times 1$, A es una matriz $m \times n$ y x es un vector $n \times 1$, y

$$\text{mín}\, w = b^T y \tag{3.17}$$

sujeto a

$$A^T y \geq c \tag{3.18}$$

$$y \geq 0 \tag{3.19}$$

donde y es un vector $m \times 1$. Cada uno de estos problemas es dual del otro.

Column Generation

Column Generation o *Delayed Column Generation* es un algoritmo destinado a resolver programas lineales de gran tamaño.

Este procedimiento se basa en la idea de la existencia de programas lineales demasiado grandes como para considerar todas sus variables de forma explícita. Debido a que la mayoría de las variables del modelo se tratarán como "no básicas", y se asume para ellas un valor de cero en las solución óptima, únicamente un subconjunto de variables serán consideradas como "básicas" para resolver el problema.

Por lo anterior, esta técnica tiene como objetivo tratar únicamente con las variables que tienen potencial para mejorar la función objetivo; esto es, buscando variables que tengan el coste reducido negativo.

El proceso resuelve el problema dividiéndolo en dos: el problema "maestro" y el subproblema correspondiente. El problema maestro es el original, pero considerando únicamente un subconjunto de variables. El subproblema es un nuevo problema creado a tal fin de identificar una nueva variable. La función objetivo del subproblema es el valor del coste reducido de la nueva variable respecto a las variables actuales.

Este algoritmo funciona de la siguiente manera: Se resuelve el problema maestro. Se obtienen valores duales para cada una de las restricciones del problema que serán utilizados en el subproblema. Se soluciona el subproblema. Si el valor objetivo del subproblema es negativo, entonces significa que se ha identificado una variable con coste reducido negativo. Esta variable es, por tanto, incorporada al problema maestro, siendo éste otra vez resuelto, y obteniéndose otra vez un nuevo conjunto de valores duales. El proceso se repite hasta que no se encuentran variables con coste reducido negativo. El subproblema devuelve una solución con coste reducido no negativo y se concluye afirmando que la solución del problema maestro es óptimo.

Algoritmo *Branch & Price*

Fundamentalmente, este proceso consiste en aplicar el método de ramificación y acotación, de tal forma que en cada nodo del árbol de búsqueda se pueden añadir columnas a la relajación LP (LR).

Al comienzo del algoritmo, conjuntos de columnas son excluidos de la LR, con el objetivo de reducir los requerimientos computacionales de procesamiento y memoria. Las columnas pueden volver a la relajación LP cuando sea necesario. Este proceso se basa en la experiencia encontrada en los problemas de grandes dimensiones. En éstos, la mayoría de las columnas no serán esenciales, teniendo su correspondiente valor a cero en la solución óptima. De esta manera, se puede concluir que la gran mayoría de las columnas son irrelevantes para resolver el problema.

El algoritmo comienza con una reformulación del problema, para crear lo que se conoce como el 'problema maestro'. Pero esta nueva formulación normalmente contiene muchas variables, por lo que se considera el uso de una versión modificada, el llamado problema maestro restringido, en el cual únicamente se tiene en cuenta un subconjunto de las columnas totales. Este nuevo problema debe ser resuelto.

Para comprobar si el resultado anterior es óptimo, se hace uso del subproblema *Pricing Problem* con el objetivo de encontrar columnas que puedan mejorar la función objetivo (por ejemplo, reducir dicha función objetivo para un problema de minimización). Esto implica buscar una columna que tenga un coste reducido negativo. Cada vez que se encuentra una columna de esta naturaleza, se incorpora al problema maestro restringido, y la relajación es reoptimizada. Si, en cambio, no se encuentran dichas columnas, y además la solución de la relajación es no entera, se debe realizar una ramificación. Si es entera, se devuelve la solución alcanzada.

Por último, si se utiliza el método de planos de corte para ajustar los resultados de la relajación, entonces el proceso global se nombraría como *Branch and Price and Cut*.

En la figura 3.5 aparece un esquema resumido del algoritmo [4].

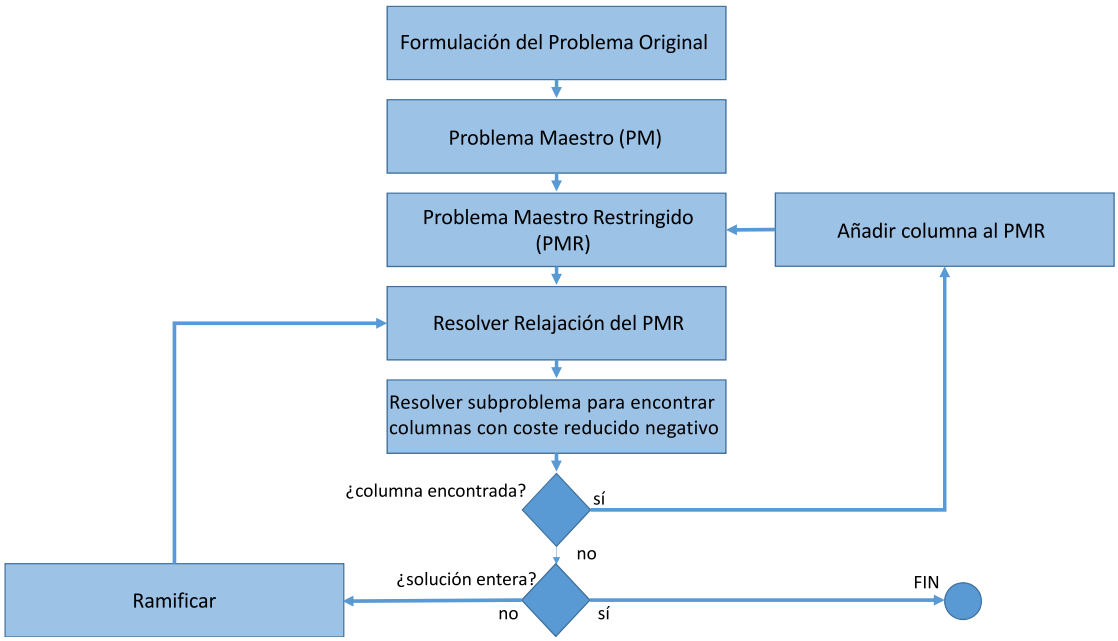

Figura 3.5: Esquema del algoritmo Branch and Price.

3.3. Metaheurísticas

3.3.1. Heurísticas

Como se comenta en [78], según el Diccionario de la Lengua Española [88], la etimología de la palabra "heurística", de origen griego, significa: "Hallar, inventar"; y entre sus definiciones se pueden encontrar las siguientes:

- "Técnica de la indagación y del descubrimiento" y
- "En algunas ciencias, manera de buscar la solución a un problema mediante métodos no rigurosos, como por tanteo, reglas empíricas, etc".

La popularización del concepto "heurística" en el ámbito de la investigación, se debe al matemático George Pólya, quien la empleó por primera vez en su libro "*How to solve it*" [192].

Asímismo, tal y como se menciona en [234], [97], un heurístico es un "procedimiento simple, a menudo basado en el sentido común, que se supone que ofrecerá una buena solución (aunque no necesariamente la óptima) a problemas difíciles, de un modo fácil y rápido".

De igual manera, el término heurística hace referencia a métodos capaces de proporcionar soluciones a problemas de optimización. Dichos procedimientos pueden ser genéricos o específicos. Los procesos genéricos suelen ser sencillos, adaptables y robustos, además de independientes del problema para el que se emplean. En cambio, los de tipo específico, suelen ofrecer soluciones de más calidad que los primeros, si bien necesitan ser particularizados para el problema en cuestión.

Una heurística puede partir de una solución previa, o bien, puede construir una solución desde el principio. Estas últimas son denominadas heurísticas constructivas, mientras que las que tratan de buscar una mejor solución, realizando movimientos a partir de un resultado ya creado, se mencionan como heurísticas de búsqueda o búsquedas locales (*Local Search* -LS-). De manera más detallada:

- Constructivo. Procedimiento heurístico capaz de crear una resultado para un problema determinado, partiendo de una solución vacía. Se van añadiendo elementos a la solución, paso a paso, hasta que la salida es completada. En cada iteración, un elemento es escogido empleando un criterio de selección.
- Búsqueda local. Proceso heurístico que parte de una solución ya creada, y trata de encontrar otra de mejor calidad. Se basa en el que, probablemente, sea uno de los procedimientos de optimización más antiguos: "prueba y error". Para ello, realiza modificaciones o movimientos dentro de la solución de partida y evalúa la calidad del nuevo resultado obtenido.

Algunos autores [208] añaden a la lista anterior, la siguiente clasificación de métodos de resolución mediante heurísticos:

- Descomposición. Dividen el problema en varios más pequeños y la solución se obtiene a partir de los resultados de cada uno de éstos.
- Reducción. Trata de identificar alguna característica de la solución que permita simplificar el tratamiento del problema.
- Manipulación del modelo. Obtiene una solución del problema original a partir de otro problema simplificado (con menos restricciones, "linealizando" el ejercicio, etc.).

Por último, hay que destacar que el principal inconveniente que presentan las heurísticas, en el área de la optimización, es su incapacidad de escapar de óptimos locales.

3.3.2. Metaheurísticas

Como se ha mencionado anteriormente, el principal inconveniente que presentan los resultados procedentes de heurísticas, es la alta probabilidad de acabar atrapado en mínimos o máximos locales, quedando, por tanto, "lejos" de los óptimos.

A raíz de esto, se define un nuevo tipo de técnicas: las "metaheurísticas". Según el Diccionario de la Lengua Española [88] el prefijo "meta-" significa "junto a", "después de", "entre", "con" o "acerca de"; si bien es común darle el sentido de "más allá de". En inglés, también está aceptado su uso como "por encima de" [222].

El término "metaheurística" como tal, fue acuñado por Fred Glover en el año 1986 [103], [78]. Con este término, pretendía definir un procedimiento maestro de alto nivel que guiase y modificase otras heurísticas para explorar soluciones más allá de los óptimos locales.

Con otras palabras, una metaheurística es un conjunto de conceptos que se emplean para definir métodos heurísticos, que pueden ser aplicados a una amplia variedad de problemas. Se considera como un marco general algorítmico, que se puede aplicar a diferentes problemas de optimización, con mínimos cambios para ser adaptado a un problema específico. También se pueden definir como procedimientos iterativos que guían una heurística subordinada, combinando de forma inteligente distintos conceptos para explorar y explotar adecuadamente el espacio de búsqueda. Son, por tanto, algoritmos aproximados de optimización y búsqueda de propósito general.

De esta manera, las técnicas metaheurísticas son capaces de proporcionar muy buenas soluciones (no necesariamente la óptima pero sí aproximada) en tiempo y con recursos razonables. Tal y como se comenta en [96], las propiedades fundamentales que caracterizan a las metaheurísticas son las siguientes [35]:

- Son estrategias que guían el proceso de búsqueda.
- Su objetivo es explorar eficientemente el espacio de búsqueda con el fin de encontrar soluciones cercanas a las óptimas.
- Las técnicas que constituyen los métodos metaheurísticos engloban desde la búsqueda local hasta procesos complejos de aprendizaje.
- Los algoritmos metaheurísticos son aproximados y usualmente no deterministas.
- Incorporan mecanismos que evitan que el proceso de búsqueda quede atrapado en áreas confinadas del espacio (óptimos locales).
- Los elementos básicos de las metaheurísticas se pueden describir a un nivel abstracto.
- No son específicas de un problema concreto.
- Pueden hacer uso del conocimiento particular del dominio de trabajo en forma de heurísticas, controladas por la estrategia de alto nivel.

A continuación se enumeran algunas de las tácticas que utilizan las metaheurísticas en el proceso de búsqueda:

- Relajación. Procedimiento de resolución de problemas que utiliza relajaciones del modelo original (modificaciones del modelo que hacen el problema más fácil de resolver), cuya salida facilita la solución del problema original.

- Construcción. Crear "desde el inicio" una solución, añadiendo en cada etapa un componente hasta completarla. A la hora de seleccionar el elemento, se suele elegir al más prometedor. Pero, en algunas ocasiones, la elección del individuo a incorporar se "aleatoriza" ligeramente, para no generar la misma solución en diferentes ejecuciones del algoritmo. En otras ocasiones, se penaliza a aquellos elementos que ya hayan sido seleccionados en ejecuciones previas.

- Búsqueda local. Proceso que utiliza transformaciones o movimientos en las soluciones para recorrer el espacio de búsqueda. Se parte de una solución inicial y, mediante una operación definida previamente, llamada movimiento, ésta se va transformando paso a paso; de manera que, en cada etapa, el nuevo resultado mejore el anterior. Algunos algoritmos de búsqueda local permiten empeorar temporalmente una solución durante el proceso de mejora, con el objetivo de escapar de óptimos locales.

- Evolución. Procedimiento basado en poblaciones o conjuntos de soluciones que van mejorando a lo largo del tiempo. Estos métodos se basan en generar, seleccionar, combinar y reemplazar un conjunto de soluciones.

En la aplicación de un procedimiento metaheurístico a un problema concreto, es habitual que se combinen diferentes tipos de estrategias, en diferentes etapas o fases del algoritmo. Por ejemplo, las metaheurísticas evolutivas suelen emplear en algunas de sus fases tácticas constructivas y de búsqueda local.

3.3.3. Intensificación y diversificación

La intensificación y la diversificación son ideas que se desarrollaron originariamente en el marco de la búsqueda tabú [104], [105] y [107]. Actualmente, se han extendido al resto de metaheurísticas como medida de su "potencial".

La intensificación es la forma de definir el proceso de búsqueda más exhaustivo que puede llevar a cabo una metaheurística en una vecindad dada. Generalmente el método no intensifica en cualquier entorno, ya que eso sería una búsqueda absoluta de todo el espacio de soluciones. Por esta razón, debe tener en cuenta cómo puede ser la calidad de las soluciones encontradas en la vecindad; y así, si en la cercanía no existe ninguna solución de calidad, no parecería interesante ni conveniente seguir con la búsqueda en esta región.

La diversificación es la forma de definir el proceso mediante el cual el algoritmo es capaz de visitar diversas vecindades lejanas. Generalmente no se diversifica constantemente y sin criterio, ya que esto correspondería con una búsqueda aleatoria en el espacio de soluciones. En este caso, se puede tener en cuenta diferentes factores, entre los que se puede encontrar la visita previa a una región dada (si se han realizado tareas de búsqueda en dicho área, no se volvería a efectuar dicha acción sobre la zona).

En la mayoría de los problemas, estas estrategias son opuestas. De tal forma que cuanto más tiempo se dedique el algoritmo a intensificar la búsqueda en una región dada, menos tiempo se podrá destinar a buscar en regiones inexploradas. También se daría el mismo caso en el otro sentido.

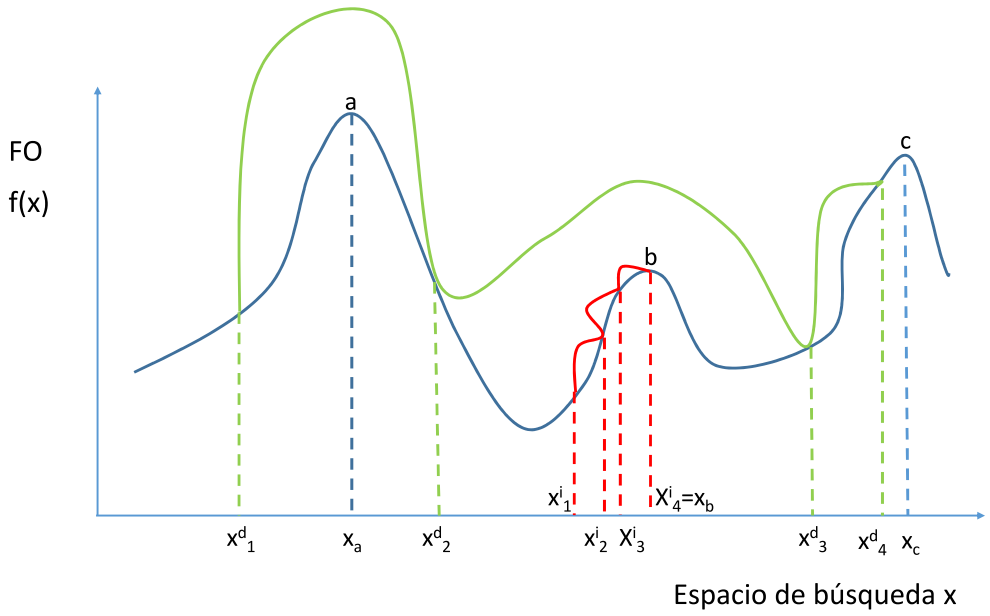

Figura 3.6: Intensificación y diversificación en un espacio de búsqueda de un problema de optimización.

Como se puede apreciar en la figura 3.6, las estrategias de intensificación y diversificación permiten explorar el espacio de soluciones de un problema de maximización de diferentes formas. Mientras que la intensificación, a partir de una solución inicial x_1^i, rastrea los vecinos x_2^i y x_3^i hasta encontrar un máximo x_4^i, que coincide con x_b (camino rojo); la diversificación trata de buscar en diferentes zonas del espacio de exploración. De esta manera, desde una solución inicial x_1^d, salta a diversas partes del espacio x_2^d y x_3^d hasta pararse en un máximo x_4^d (camino verde). Ni x_4^i ni x_4^d contienen el mejor resultado del espacio de soluciones. El primero se encuentra en un máximo local (b), lo que impide llegar a una mejor solución al seguir "intensificando". En el segundo caso, se obtiene un mejor resultado, pero inferior al que se conseguiría en los puntos x_a (con la diversificación) o x_c (aplicando intensificación). Parece lógico pensar que una hibridación de las dos técnicas permitiría obtener mejores resultados que aplicando éstas por separado.

3.3.4. Metaheurísticas trayectoriales

Estas técnicas parten de un punto inicial y van actualizando la solución presente mediante la exploración del vecindario, formando una trayectoria. La búsqueda finaliza cuando se alcanza o bien un número máximo de iteraciones, o bien se encuentra una solución con una calidad aceptable; o se detecta un estancamiento del proceso.

A continuación se comentan algunos procedimientos trayectoriales tratados o mencionados en este documento en algunos de sus apartados.

Recocido simulado

Como se comenta en [35], se dice que *Simulated Annealing* (SA) es el más antiguo de entre las metaheurísticas, y seguramente uno de los primeros algoritmos con una estrategia explícita para escapar de los óptimos locales.

La idea fundamental de este método se basa en la aceptación de movimientos que conducen a peores resultados que el actual, con el objetivo de evitar los mínimos o máximos locales. La probabilidad de llevar a cabo esta acción se decrementa conforme va avanzando la búsqueda. El pseudocódigo de alto nivel para un problema de minimización aparece en el algoritmo 3.4.

El proceso comienza generando una solución inicial (obtenida de forma aleatoria o heurística), así como iniciando el parámetro temperatura T. A continuación, en cada iteración se toma aleatoriamente una solución $s' \in N(s)$ del vecindario de la solución actual s. Su aceptación dependerá de los valores $f(s)$, $f(s')$ y T (siendo f la función objetivo del problema). En caso de resolver un ejercicio de minimización, s' será la nueva solución, y reemplazará a s como solución actual si $f(s') < f(s)$. En caso contrario; es decir, $f(s') \geq f(s)$, s' sustituirá a s con una probabilidad p que es función de t y $f(s') - f(s)$. Dicha probabilidad normalmente se expresa según la distribución de Botzmann:

$$p = \mathrm{e}^{-\frac{f(s') - f(s)}{T}} \tag{3.20}$$

La temperatura T se decrementa durante el proceso de búsqueda. De tal forma que gradualmente va reduciendo su valor.

Algoritmo 3.4: `Simulated Annealing`

```
Input : T, s {s, initial solution}
Output : s {s, the procedure outcomes s}
 1: L ← add(L, ILP) {Set initial temperture}
 2: while not endConditions() do
 3:    s ← pickRandomly(N(s)) {N(s) = neighborhood of s}
 4:    if f(s') < f(s) then
 5:       s ← s' {s'replaces s }
 6:    else
 7:       {f(s') ≥ f(s)}
 8:       s ← P(T, s', s) {it is accepted as a new solution with P probability }
 9:    end if
10:    update(T) {T is decreasing in following iterations}
11: end while
12: return s
```

Busqueda tabú

Tabu Search (TS) se encuentra entre las metaheurísticas más utilizadas en la resolución de problemas de optimización [35]. Las ideas básicas del método se introdujeron en [103], basadas en las ideas preliminares de [102]. Una descripción del método y sus conceptos se encuentra en [107].

Algoritmo 3.5: `Tabu Search`

```
Output : s {solution}
1: s ← generateInitialSolution()
2: TabuList ← ∅
3: while not endConditions() do
4:    {take the best solution from the neighborhood, that isn't in the tabu list}
5:    s ← getTheBest(N(s))   /   s ∉ TabuList
6:    update(Tabulist, s) {add s to the tabu list}
7: end while
8: return s
```

El método TS simple (ver pseudocódigo 3.5) aplica una búsqueda local como base de sus acciones, así como una "memoria" tanto para escapar de los óptimos locales, como para evitar ciclos. Esta estructura algorítmica se implementa como una "lista tabú", que almacena las últimas soluciones recientes, para evitar volver a buscar en ellas. La búsqueda en el vecindario de la solución actual $N(s)$ se restringe a un elemento que no se encuentre en la lista tabú. En cada iteración, el mejor resultado obtenido, de entre las soluciones permitidas, se convierte en la mejor solución actual, añadiéndose, a continuación, a la lista tabú. Para completar la acción anterior, se debe eliminar alguna de las soluciones que se encuentren previamente en dicha lista (normalmente es una estructura FIFO, por lo que se eliminaría el individuo más antiguo de la lista). El procedimiento finaliza cuando se cumplan las condiciones de terminación, que podría darse cuando todos los elementos de $N(s)$ se encuentren en la lista tabú.

Algoritmo 3.6: `Iterated Local Search`

```
Output : s' {solution}
1: s_0 ← generateInitialSolution()
2: s' ← localSearch(s_0)
3: while not endConditions() do
4:    s'' ← generatePerturbation(s')
5:    s''' ← localSearch(s'')
6:    s' ← achieveAcceptationCriteria(s'', s''')
7: end while
8: return s'
```

Búsqueda local iterativa

Iterated Local Search (ILS), tratado en [211], [212], [156], [157] y [164], básicamente aplica la búsqueda local a una solución inicial hasta que encuentra el óptimo local. Entonces, "perturba" la solución y reinicia la búsqueda local. Como se puede apreciar, la importancia de la fase de perturbación es obvia: con una alteración reducida, esta tarea no es capaz de alejar la solución de la influencia del óptimo local encontrado (no empleando, por tanto, una buena diversificación). Por otra parte, una perturbación desmedida, convertiría el algoritmo en una búsqueda local con reinicio aleatorio.

En el algoritmo 3.6 aparecen los pasos principales del proceso. Una vez obtenida una solución inicial s_0, se aplica una búsqueda local con la salida s'. Ésta se va a alterar, obteniendo s'', que

se vuelve a enviar a un procedimiento de búsqueda local con el resultado s'''. Si no se cumple con las condiciones de aceptación, se debe volver a iterar.

En la figura 3.7 aparece un ejemplo del uso de este método en un problema de minimización. Una vez que la solución se encuentra en un mínimo local s', se produce una perturbación de la solución anterior, "saltando" a s''. A partir de aqui, se debe seleccionar el proceso de búsqueda local para refinar la solución anterior, y alcanzar, finalmente, s'''.

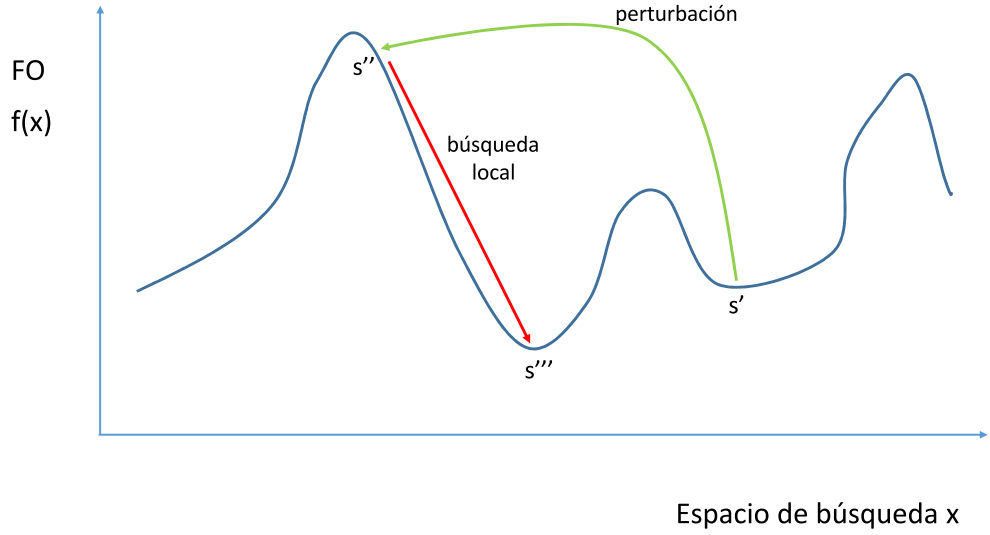

Figura 3.7: Ejemplo de búsqueda local iterativa.

Algoritmo voraz multiarranque

MSGA (*Multi-start Simple Greedy Algorithm*) es un algoritmo multiarranque que contiene un método voraz como procedimiento constructivo de soluciones.

Algoritmos voraces Un algoritmo voraz (también conocido como ávido, devorador o goloso) es aquel que, para resolver un determinado problema, sigue una heurística consistente en elegir la opción óptima en cada paso local, con la esperanza de llegar a una solución general óptima. Los algoritmos voraces constituyen estrategias sencillas que se emplean para resolver diversos problemas de optimización [38] (p. ej. caminos mínimos en grafos, planificación de tareas, etc.). Una solución se construye incrementalmente, incorporando en cada paso a un elemento candidato que mantenga la factibilidad de la solución. Inicialmente, el conjunto de candidatos se ordena según cierto(s) criterio(s). En general, las estrategias voraces permiten conseguir buenas soluciones con un tiempo de cómputo razonable en algunos problemas.

Una vez definidos los procedimientos voraces, se puede destacar las características más importantes del método MSGA:

- Va a utilizar un algoritmo voraz para construir las soluciones del problema.
- Por otra parte, al ser éste un método multiarranque, intenta evitar que el resultado quede atrapado en óptimos locales, reiniciando nuevamente el procedimiento.

Algoritmo 3.7: `Multi-start Simple Greedy Algorithm`

```
Output : bestSolution
1: bestSolution ← ∅
2: permutations ← generatePerm()
3: for i = 1 to size(permutations) do
4:    currentPermutation ← getAtRandom(permutations)
5:    solution_i ← greedyAlgorithm(currentPermutation)
6:    if objectiveFunct(solution_i) > objectiveFunct(bestSolution) then
7:       bestSolution ← solution_i
8:    end if
9: end for
10: return bestSolution
```

En el algoritmo 3.7 aparece el pseudocódigo a alto nivel del metaheurístico para un problema de maximización. Hay que destacar que para introducir diversidad en la exploración, en cada iteración del método, el algoritmo voraz tomará diferentes propiedades de rastreo de forma aleatoria. Estas características se generan y almacenan en la estructura `permutations`.

3.3.5. Metaheurísticas poblacionales

Las metaheurísticas basadas en poblaciones o poblacionales se caracterizan por trabajar en cada iteración del algoritmo con un conjunto de soluciones denominado "población". Estos métodos proporcionan de forma intrínseca un mecanismo de exploración paralela en el espacio de soluciones. Por tanto, la eficiencia del proceso dependerá fundamentalmente de la creación y gestión de la población. A continuación, se describen algunos procedimientos de esta naturaleza.

Algoritmo de la colonia de hormigas

Los algoritmos basados en *Ant Colony Optimization* (ACO) son un tipo de metaheurística poblacional, cuya filosofía se encuentra inspirada en el comportamiento de las hormigas reales cuando buscan comida [75], [54], [35]. La idea principal consiste en emplear hormigas artificiales que simulan dicho comportamiento en un escenario también artificial: un grafo.

La hormiga artificial Cada hormiga artificial (un agente que imita a la hormiga natural) es un mecanismo probabilístico de construcción de soluciones al problema que utiliza los siguientes datos:

- Unos rastros de feromona (artificiales) τ que cambian con el tiempo, para reflejar la experiencia adquirida por los agentes en la resolución del problema.
- Información heurística η sobre la instancia concreta del problema (por ejemplo, la longitud del camino).

Asímismo, utiliza una regla probabilística de transición. La más habitual define la probabilidad de que la hormiga k, situada en el nodo i del grafo, decida moverse hacia el nodo j por medio

del arco $a_{i,j}$:

$$p_k(i,j) = \begin{cases} \dfrac{[\tau_{i,j}]^\alpha \cdot [\eta_{i,j}]^\beta}{\sum_{u \in L_k(i)} [\tau_{i,j}]^\alpha \cdot [\eta_{i,j}]^\beta} & \text{si } j \in L_k(i) \\ 0 & \text{en otro caso} \end{cases} \tag{3.21}$$

donde:

$\tau_{i,j}$ es la feromona del arco $a_{i,j}$.

$\eta_{i,j}$ es la información heurística del arco $a_{i,j}$.

α y β son pesos que establecen la importancia relativa entre los niveles de feromona y la información heurística, respectivamente.

$L_k(i)$ es el conjunto de nodos alcanzables desde i no visitados aún por la hormiga k.

La metaheurística de optimización basada en colonias de hormigas (OCH) De forma general, las m hormigas (artificiales) de la colonia se mueven, concurrentemente y de manera asíncrona, a través de los estados adyacentes del problema (que puede representarse en forma de grafo). Este movimiento se realiza siguiendo una regla de transición, que está basada en la información local disponible en las componentes (nodos). Esta información incluye datos heurísticos y memorísticos (rastros de feromona), empleados para guiar la búsqueda. Al moverse por el grafo, las hormigas construyen incrementalmente soluciones.

El Sistema de Hormigas (SH o *Ant System* AS-), desarrollado por Dorigo, Maniezzo y Colorni en 1991, fue el primer algoritmo de esta naturaleza. Un pseudocódigo se presenta en el algoritmo 3.8.

Algoritmo 3.8: `Ant System`

```
Input : parameters
Output : bestSolution
{Initialize parameters}
gen ← initializePheromones()
bestSolution ← initializeSolution()
Ants ← initializeAnts()
repeat
  for k = 1 to size(Ants) do
    solution_k ← findSolution(Ants[k])
  end for
  update(Pheromones)
  bestSolution ← updateBestSol(bestSolution, solution_i)/i = 1, ..., size(Ants)
until k ≥ MAX_NUM_ITERATION or otherTerminationConditions()
return bestSolution
```

Algoritmos evolutivos

Tal y como se describe en [35], la computación evolutiva se inspira en la capacidad de los seres vivos de adaptarse a su entorno. Los algoritmos evolutivos (EA) aplican a los individuos de la población una serie de operaciones para generar nuevos individuos en las siguientes iteraciones.

Mediante la operación de cruce o *crossover*, se recombinan dos o más individuos para generar unos nuevos; la "mutación" causa una auto-adaptación del individuo al entorno; y la operación de "selección" escoge a los individuos según su aptitud (mediante alguna función objetivo, por ejemplo). Los individuos con mayor calificación tendrán mayor probabilidad de ser escogidos en la siguiente iteración. El pseudocódigo aparece en el algoritmo 3.9.

Algoritmo 3.9: `Evolutionary algorithm`

```
1: while not terminationCondition() do
2:    P' ← crossover(P)
3:    P'' ← mutation(P')
4:    evaluation(P'')
5:    P ← P ∪ P
6: end while
```

A continuación, se tratan los algoritmos genéticos (GA), que se enmarcan dentro de los EA.

Algoritmo genético El algoritmo genético o GA (*Genetic Algorithm*) es una metaheurística poblacional sencilla e intuitiva. Se basa en una población de soluciones candidatas que evoluciona a través de los mecanismos genéticos neo-darwinistas de selección, cruce y mutación. Fue ideada por Holland en 1975 [118] y está inspirada en patrones de adaptación y evolución de los seres vivos: por un lado, estos patrones permiten que con el transcurso del tiempo, se exploren continuamente nuevas posibilidades; y en condiciones normales, raramente conducen a la obtención de individuos absolutamente desadaptados e incapaces de sobrevivir.

Algoritmo 3.10: `Genetic Algorithm`

```
Input : P {set of individuals (population)}
Output : bestSolution
 1: gen ← 0 {gen is the current iteration}
 2: P(gen) ← initializePopulation() {generate initial set of individuals}
 3: F(gen) ← evaluatePopulation(P(gen)) {F measures the quality of each individual}
 4: repeat
 5:    gen = gen + 1
 6:    {Selection (select better individuals) and crossover operations}
 7:    P(gen) ← selectAndCrossover(P(gen − 1), F(gen − 1)
 8:    {Mutation operation}
 9:    P(gen) ← mutation(P(gen))
10:    {Evaluate the new population}
11:    F(gen) ← Evaluate(P(gen))
12: until endConditions()
13: return bestSolution ← getBestSolution(P)
```

Partiendo de una población inicial, es decir, un conjunto inicial de soluciones, se realizan manipulaciones por las que se obtienen sucesivas poblaciones. La función de adaptación indica la bondad de las soluciones consideradas en cada momento.

En cada iteración se realizan una serie de operaciones con los individuos de la población, de entre las cuales las más comunes son: la selección, el cruce, la mutación y la inversión. La

aplicación de los operadores anteriores permite obtener, típicamente, soluciones con mejores funciones de adaptación.

Los algoritmos genéticos pertenecen al grupo de las técnicas evolutivas, que son aquellas que en cada iteración disponen de un conjunto de soluciones a partir de las cuales se obtienen un nuevo conjunto de resultados.

Los GA han introducido una notación con una marcada interpretación biológica, que en algunos casos se ha extendido al resto de metaheurísticas. En el contexto de estos algoritmos, se dice que un conjunto de soluciones es una población de individuos, de forma que cada individuo se corresponde con una solución. La solución con las variables del problema establece el "fenotipo" del individuo; y su representación el "genotipo". Esta última recibe normalmente el nombre de "cromosoma". Cada uno de éstos suele estar compuesto por unidades llamadas "genes", y el valor de cada gen se conoce como "alelo". A continuación se detallan los elementos que se deben identificar en un algoritmo de este tipo:

- Población inicial. Generalmente consiste en una generación aleatoria de soluciones al problema dado. Se puede utilizar una heurística o metaheurística para construir una población inicial de calidad.

- Representación. Es la correspondencia entre las soluciones factibles (fenotipo) y la codificación de las variables (genotipo).

- Función de evaluación. Determina la calidad de cada uno de los individuos que pertenecen a la población.

- Operadores genéticos. Son métodos probabilísticos que no tienen por qué aplicarse a toda la población. Suelen ser los siguientes:

  - Cruce (*crossover*). Consiste en la sustitución de un conjunto de genes de un padre por los genes correspondientes de otro padre para generar cromosomas descendientes.

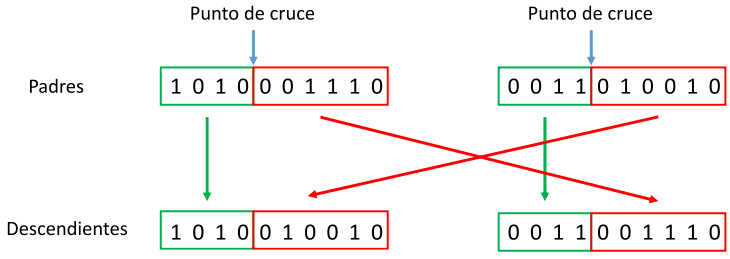

Figura 3.8: Operación de cruce en un GA.

  - Mutación. Consiste en el cambio aleatorio de un alelo. En general, la mutación permite preservar la diversidad de las soluciones, así como la posibilidad de explorar nuevas zonas en el espacio de búsqueda.

- Selección. Permite seleccionar con mayor probabilidad a los individuos que presenten un valor más elevado de la función de evaluación (intensificación).

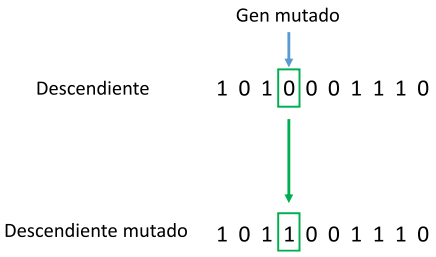

Figura 3.9: Operación de mutación en un GA.

Es importante destacar que la cantidad de variantes que se pueden dar en estos algoritmos es enorme. En este apartado, únicamente se ha indicado las caractersticas básicas. En el algoritmo 3.10 se muestra el pseudocódigo para un algoritmo evolutivo general, y genético en particular.

Algoritmo de enjambre de partículas

Los algoritmos basados en nubes (también enjambre o cúmulos) de partículas PSO (*Particle Swarm Optimization*) son una técnica metaheurística poblacional que se inspira en la naturaleza para resolver problemas (algoritmo bioinspirado). En concreto, en el comportamiento social del vuelo de las bandadas de aves o el movimiento de los bancos de peces. Es una técnica originalmente desarrollado por el sociólogo James Kennedy y por el ingeniero Russ C. Eberhart en 1995 [81], [132].

Estos autores describen el algoritmo PSO de la siguiente manera: los individuos (partículas) que conviven en una sociedad tienen una "opinión" que es parte del espacio de búsqueda, compartido por todos los individuos. Cada individuo puede modificar su opinión según tres factores:

- El conocimiento del entorno o adaptación.
- Experiencias anteriores del individuo o memoria del individuo.
- Experiencias anteriores de los individuos del vecindario o memoria del vecindario.

PSO es un sistema multiagente. Las partículas son agentes simples que se mueven por el espacio de búsqueda; guardan y posiblemente comunican la mejor solución que han encontrado. El movimiento de las partículas por el espacio está guiado por los individuos que manejan la mejor solución del momento.

Algoritmo Se inicia la descripción del procedimiento PSO estudiando la anatomía de la partícula. Ésta está compuesta por tres vectores y dos valores de aptitud (también conocida como bondad, adaptación, capacidad, adecuación o *fitness*) con respecto al problema considerado [108]. Para cada partícula i se tiene los siguientes elementos:

- Tres vectores:
  - El vector $x_i = (x_{i,1}, x_{i,2}, ..., x_{i,N})$ almacena la posición actual de la partícula i.

- El vector $mejorpos_i = (mejorpos_{i,1}, mejorpos_{i,2}, ..., mejorpos_{i,N})$ almacena la posición de la mejor solución encontrada por la partícula i hasta el momento.
- El vector de velocidad $v_i = (v_{i,1}, v_{i,2}, ..., v_{i,N})$ almacena la dirección según la cual se moverá la partícula i.

- Dos valores de aptitud:
  - El valor de $aptitud_x_i$ almacena el valor de adaptación o adecuación de la posición actual correspondiente al vector x_i.
  - El valor de $aptitud_mejorpos_i$ guarda el valor de adecuación de la partícula i con mejor solución local encontrada hasta el momento, correspondiente al vector $mejorpos_i$.

La descripción del proceso algorítmico es la siguiente:

1. La nube se inicializa generando las posiciones (de forma aleatoria, regular o combinación de ambas).
2. Se generan las velocidades de forma aleatoria en un intervalo establecido $[-v_{max}, v_{max}]$. No es conveniente fijarlas a cero.
3. Se calcula la aptitud de cada partícula y se actualizan los valores de $aptitud_x_i$ y $aptitud_mejorpos_i$.
4. Las partículas se mueven en cada iteración desde una posición del espacio de búsqueda hasta otra. Al vector de posición x_i se le añade el de velocidad v_i para obtener un nuevo vector x_i.
5. Con la nueva posición de la partícula se calcula y actualiza $aptitud_x_i$.
6. Si el nuevo valor de aptitud es el mejor encontrado por la partícula i hasta el momento, se actualizan los valores de $mejorpos_i$ y $aptitud_mejorpos_i$.
7. Si el nuevo valor de $aptitud_mejorpos_i$ es el mejor encontrado por la nube de partículas hasta el momento, se actualizan tanto el valor de la mejor posición de la nube $mejorpos$ y su $aptitud_mejorpos$.
8. El vector velocidad de cada partícula es modificado en cada iteración utilizando la velocidad anterior, un componente cognitivo y un componente social. El modelo matemático resultante, y que representa el "núcleo" del algoritmo, viene representado por las siguientes ecuaciones:

$$v_i^t \leftarrow \omega^{t-1} \cdot v_i^{t-1} + \varphi_1 \cdot rand_1 \cdot (mejorpos_i - x_i^{t-1}) + \varphi_2 \cdot rand_2 \cdot (mejorpos - x_i^{t-1}) \quad (3.22)$$

$$x_i^t \leftarrow x_i^{t-1} + v_i^t \quad (3.23)$$

para $i = 1, 2, ..., P$, donde P es el número de partículas que componen la nube:

- x_i^t y v_i^t son los vectores posición y velocidad de la partícula i en la iteración t.
- ω_t es el factor de inercia en la iteración t.
- φ_1, φ_2 son pesos que controlan los componentes cognitivo y social.
- $rand_1$ y $rand_2$ son números aleatorios entre 0 y 1.
- $mejorpos_i$ es la mejor posición encontrada por la partícula i hasta el momento.
- $mejorpos$ representa la posición de la partícula con la mejor solución o aptitud.

Componentes inercial, cognitivo y social La ecuación 3.22 actualiza el vector velocidad de cada partícula i en la iteración t. El primer término de la ecuación $\omega^{t-1} \cdot v_i^{t-1}$, que incluye el vector velocidad de la anterior iteración, es el componente inercial. El componente cognitivo muestra la decisión que tomará la partícula y depende de su propia experiencia. Dicho de otra manera, representa la distancia entre la posición actual y la mejor conocida por esa partícula. Sería el factor siguiente: $\varphi_1 \cdot rand_1 \cdot (mejorpos_i - x_i^{t-1})$. El componente social marca la decisión que tomará la partícula en base a la influencia del resto de partículas que componen la nube. Es decir, representa la distancia entre la posición actual y la mejor posición encontrada en el vecindario. El componente social se modela en la ecuación de la siguiente forma: $\varphi_2 \cdot rand_2 \cdot (mejorpos - x_i^{t-1})$.

La fórmula 3.23 calcula el vector de posición x_i de la partícula i para cada iteración, teniendo en cuenta la velocidad v_i^t calculada previamente.

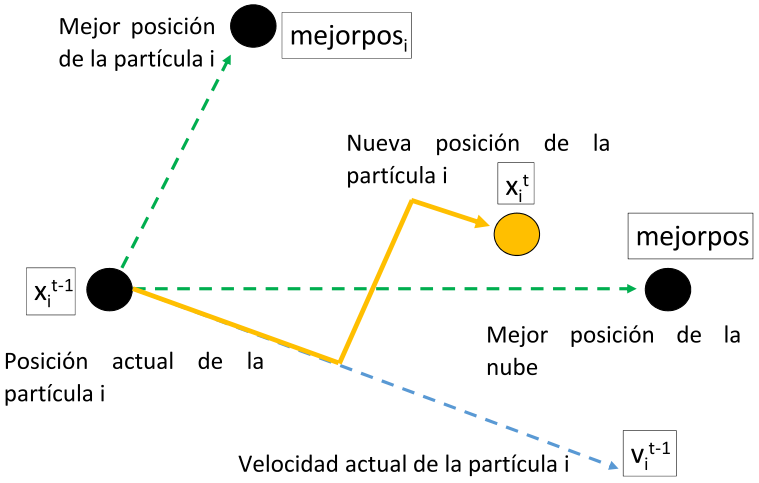

Figura 3.10: Representación gráfica del movimiento de una partícula.

La figura 3.10 muestra el movimiento de una partícula en el espacio de soluciones. Las flechas de color verde discontinua representan la dirección de los componentes cognitivo y social. La flecha azul discontinua representa la velocidad actual de la partícula (componente inercial). La flecha de línea continua representa la dirección que toma la partícula para moverse desde la posición actual (x_i^{t-1}) hasta la nueva posición x_i^t.

Algoritmo 3.11: `Particle Swarm Optimization`

```
Output : bestPos, aptitudeBestPos
Swarm ← initialize()
while not terminationCondition() do
  for i = 1 to size(Swarm) do
    (pos_x_i, aptitude_x_i) ← evaluate(x_i) {x_i is a swarm's particle number i}
    if aptitude_x_i > aptitude_bestPos_i then
      bestPos_i ← x_i
      aptitude_bestPos_i ← aptitude_x_i
    end if
    if aptitude_bestPos_i > aptitude_bestPos then
      bestPos ← bestPos_i
      aptitude_bestPos ← aptitude_bestPos_i
    end if
  end for
  for i = 1 to size(Swarm) do
    v_i ← calculateVelocity(x_i, bestPos_i, bestPos)
    x_i ← calculateVelocity(x_i, v_i)
  end for
end while
return (bestPos, aptitudeBestPos)
```

El algoritmo PSO trabaja de forma iterativa modificando una nube de partículas mediante la aplicación de movimiento a cada una de ellas. El pseudocdigo 3.11 describe el algoritmo PSO clásico para un problema de maximización. Las variantes a este algoritmo dependen fundamentalmente de la implementación de las partículas, el cálculo del vector velocidad y del movimiento de las partículas, y la representación de las soluciones.

3.3.6. Comparación de metaheurísticas

Con todo el abanico de técnicas metaheurísticas que existen, parece claro que es conveniente elegir el procedimiento más apropiado al problema a resolver, con el objetivo de poder tener éxito con el resultado final del algoritmo elegido. La propuesta seleccionada debe ser aquella que se adapte mejor a la naturaleza del ejercicio a solucionar de una manera más sencilla. No parece ésta una tarea trivial. Ms bien, va a ser la experiencia en estas técnicas, la que guíe el desarrollo de futuras soluciones metaheurísticas.

No Free Lunch Theorem en la búsqueda y la optimización

Para un problema en particular, diferentes algoritmos pueden obtener dispares resultados. Pero si la sentencia anterior intenta generalizarse para todos los problemas, los resultados globales de los diversos procedimientos serían indistinguibles. Es decir, que si un método de resolución es superior a otros en algunas clases de problemas, debe pagar un coste de inferioridad en otros problemas distintos a los primeros. Esto es lo que se conoce como *No Free Lunch Theorem* (NFL) en la búsqueda y la optimización [227], [228].

Este teorema indica que dos algoritmos cualesquiera son equivalentes en el rendimiento global de todos los posibles teoremas. Una consecuencia de la sentencia anterior es que hacer

una relación o correspondencia de los algoritmos con los problemas a resolver, va a dar mejores resultados globales que utilizar un mismo procedimiento para todos los ejercicios a solucionar. En [122] y [82] se establecieron las condiciones de tipo general bajo las cuales se cumple este teorema. [76] prueba que en la práctica se cumple el *(Almost) No Free Lunch* (ANFL). ANFL indica que no se puede esperar buenos resultados en escenarios en los que no se tiene un buen conocimiento de la función objetivo del problema; en cambio, si el procedimiento es capaz de captar correctamente las "pistas" o características que aparecen en la función del ejercicio, el método puede obtener mejores resultados que otros procesos con diferentes conjeturas.

3.4. Matheurísticas

3.4.1. Introducción

Por 2009, en [127], ya se mostraba la gran importancia que estaba cobrando los métodos matheurísticos. Éstos se podrían describir como una cooperación entre métodos exactos y heurísticos/metaheurísticos con el objetivo de resolver problemas de optimización.

En ejercicios con instancias pequeñas, la utilización de la programación exacta suele dar buenos resultados, aunque el problema sea de complejidad exponencial $O(c^{\mathrm{n}})$. Entre estos métodos se podrían incluir los siguientes: ramificación y poda (*Branch and Bound*), programación lineal, entera y dinámica.

Pero cuando las instancias empiezan a aumentar de tamaño, el problema se puede convertir en intratable para los métodos anteriores; debido principalmente a la cantidad de tiempo de ejecución necesario para resolver el problema. En [79] también se presentan otros inconvenientes adicionales como puede ser la cantidad de memoria necesaria para llevar a cabo dicha resolución, o la dificultad en el modelado de algunos problemas. Pero también exhiben algunas ventajas, como pueden ser las siguientes:

- Si el algoritmo tiene éxito en su ejecución, se obtiene el resultado óptimo.
- En otro caso, si el algoritmo se detiene antes de completar su espacio de búsqueda, se pueden obtener cotas superiores o inferiores de la solución.
- Estos métodos permiten podar partes del espacio de búsqueda en las que no se pueden encontrar soluciones óptimas.

Como se ha comentado en líneas anteriores, cuando las instancias del problema y el espacio de búsqueda empiezan a aumentar, los métodos exactos no son adecuados para resolver los problemas de optimización. En estos casos, se suelen utilizar los métodos heurísticos y metaheurísticos. Entre los primeros se suelen incluir la búsqueda local o los algoritmos voraces (*greedy*). Entre los segundos, el recocido simulado, la búsqueda tabú, los algoritmos genéticos (GA), la optimización por colonias de hormigas (ACO) o la optimización por enjambres de partículas (PSO). En [79] también se comentan las características principales de estos procedimientos. Entre los méritos se incluyen las siguientes:

- La experiencia en el empleo de estos métodos, indica que para una gran cantidad de problemas son los que alcanzan los mejores resultados.

- Permiten examinar una gran cantidad de posibles soluciones en un corto espacio de tiempo.
- Tienen mayor flexibilidad de adaptación a diferentes variantes de un problema.
- Y finalmente, de manera general, tienden a ser más fáciles de entender e implementar, respecto a los métodos exactos.

Sin embargo, también existen desventajas, como las que se enumeran en las siguientes líneas:

- No pueden asegurar la optimalidad en la solución.
- Existe dificultad para reducir el espacio de búsqueda.
- No se suele tener de manera bien definida el criterio de parada del algoritmo.
- En ciertos casos, aparece la imposibilidad de encontrar una convergencia hacia la solución óptima. En tal caso, el resultado calculado podría situarse "lejos" de la mejor solución posible.

Con todo lo anterior, y conociendo las características de ambas familias de técnicas de resolución, exactas por una parte y heurísticas y metaheurísticas por otra, se podra deducir que la colaboración entre ellas podría proporcionar un método híbrido que englobe las ventajas y fortalezas de cada uno de ellas.

Clasificaciones Se han encontrado en la literatura diferentes organizaciones de estos métodos híbridos. Por un lado, existen algunas de ellas en la que se restringe el metaheurístico a una búsqueda local. Aunque también se han desarrollado algunas otras más ambiciosas, en las que se podrían incluir cualquier método exacto o metaheurístico.

3.4.2. Combinación de métodos exactos y búsqueda local como metaheurístico

En [79] se realiza una clasificación de diferentes estrategias aplicadas a la matheurísticas; pero, en este caso, se basa en un caso particular de la programación entera como método exacto, y búsqueda local como procedimiento heurístico. A continuación se enumeran las diferentes casos encontrados:

- Uso de algoritmos exactos para explorar la vecindad en la búsqueda local.

  El algoritmo exacto resolvería algunos subproblemas de programación lineal, definidos con el objetivo de reducir el espacio de búsqueda en la inspección de la vecindad en el algoritmo de búsqueda local. Los subproblemas serían relajaciones del modelo inicial de programación entera. En [223] se aplica esta estrategia para resolver el problema de la mochila 0-1 múltiple (multidimensional) (*0-1 Multidimensional Knapsack Problem* MDKP). El método propuesto tiene como meta que las soluciones obtenidas en estos problemas relajados se encuentren relativamente cerca de la solución óptima del problema original. Los modelos de LP pueden ser resueltos mediante un algoritmo simplex, entre otros [107]. Como conclusión, se podría comentar que éste es un ejemplo de cómo se

puede reducir el espacio de búsqueda a explorar respecto a otros métodos como el *Branch & Bound*.

En [42] y [43] también se utiliza esta estructura algorítmica para resolver el problema del viajante *Travelling Salesman Problem* (TSP), y su versión asimétrica *Asymmetric Travelling Salesman Problem* (ATSP), respectivamente.

- Llevar a cabo diferentes ejecuciones del heurístico, y utilizar las mejores soluciones obtenidas para la definición de un subproblema, que será resuelto por el algoritmo exacto; y será una solución del problema original. En [14] se resuelve el problema del viajante (TSP) en dos pasos:

  1. En el primero de ellos, se ejecuta un algoritmo de búsqueda local (ILS) del tipo *Chained Lin-Kernighan* (CLK) [14], [165] o *Iterated Helsgaun* (IH) [117] un número de veces, almacenando los mejores caminos-solución obtenidos en un conjunto $\mathcal{T}$.
  2. A continuación se resuelve el problema TSP en un subgrafo que contenga los mismos vértices que el original, pero restringido a los enlaces que aparezcan al menos una vez en algunas de las rutas almacenadas en $\mathcal{T}$. De manera más formal, si el grafo original es $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, donde $\mathcal{V}$ es el conjunto de nodos, y $\mathcal{E}$ el correspondiente a los enlaces, el grafo reducido se podría describir de la siguiente forma: $\mathcal{G}' = (\mathcal{V}, \mathcal{E}')$ donde $\mathcal{E}' = \{e \in \mathcal{E} : \exists t \in \mathcal{T}, a \in t\}$.

- Dedicación de algoritmos exactos como procedimientos específicos en metaheurísticos híbridos, utilizando la información obtenida de los modelos de programación entera para guiar a la búsqueda local. Un caso consistiría en el empleo del método exacto en la etapa de perturbación del heurístico. La perturbación en un ILS introduce cambios en la solución actual que permite a la heurística escapar de los óptimos locales, pero manteniendo, de igual forma, la calidad de la solución obtenida hasta ese momento. Una idea para obtener una buena estrategia de perturbación es definir un subproblema, similar al problema original, y resolverlo mediante un algoritmo exacto o heurístico apropiado. En [157] aplican el método anterior en el *Job-Shop Scheduling Problem* (JSSP): en este problema, cada trabajo consiste en una secuencia de operaciones, cada una de ellas con un tiempo de procesamiento, que deben ser llevados a cabo por un conjunto de máquinas en un cierto orden. El objetivo final es minimizar el tiempo del último trabajo; teniendo en cuenta que existen ciertas restricciones de capacidad (una máquina no puede trabajar en dos operaciones a la vez). Para ello, resuelven subproblemas aplicando el método exacto de tipo *Branch and Bound* llamado algoritmo de Carlier [46] sobre una máquina, resolviendo el problema de planificación para dicha máquina.

- Aplicación de cotas inferiores en heurísticos constructivos. Por último, se pueden utilizar algoritmos exactos cuya finalidad es determinar las cotas inferiores en la ejecución de heurísticos constructivos. Con la información de la cota, el heurístico puede determinar la conveniencia o no de aplicar tareas de intensificación sobre una solución. Se ha aplicado esta estrategia en algoritmos de colonias de hormigas (ACO); y más específicamente, en el caso particular de *Approximate Nondeterministic Tree Search algorithm* (ANTS) [160]:

previo al comienzo del cálculo de la solución por parte de ANTS, se ejecuta el algoritmo *Gilmore-Lawler Lower Bound* (GLB) [101] [151], que resuelve el problema relajado de asignación lineal. Sobre esta última estrategia, se podría indicar la utilidad de su uso también en el cálculo de cotas superiores. Como se han comentado en líneas anteriores, con esta información se podrían reducir las acciones de intensificación sobre una solución dada. Asísmismo, se puede determinar la calidad de dicha solución en base a la cota obtenida.

La clasificación expuesta en los puntos anteriores, da una idea de los beneficios de la colaboración entre métodos exactos y heurísticos, cuya meta es mejorar la resolución de diferentes problemas como el MDKP, TSP, o el JSSP; pero parece muy enfocada a unos heurísticos determinados como puede ser el ILS. Sería adecuado mostrar otra clasificación más general, con el objetivo de dar a conocer las posibilidades más amplias que presentan este tipo de algoritmos híbridos.

3.4.3. Combinación de métodos exactos y metaheurísticos: organización colaborativa e integradora

En esta clasificación, propuesta en [196], aparecen dos clases principales. Por una lado, una relación colaborativa entre procedimientos, en la que un proceso proporciona información a otro. Y por otra, una visión integradora, en la que uno de los procesos se encuentra incluido en el otro. A continuación, se desarrollará con más detalle estas categorías.

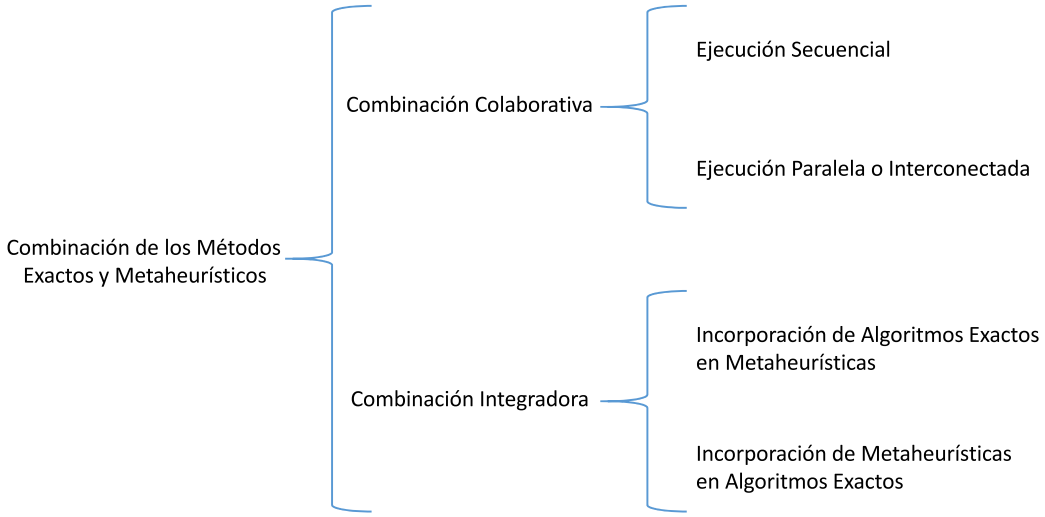

Figura 3.11: Clasificación general de las Matheurísticas.

En la figura 3.11 [196] se muestra una visión general de las diferentes posibilidades que existen en la combinación de métodos exactos y heurísticos para la resolución de problemas. Como se puede apreciar, estos algoritmos híbridos se pueden dividir según las siguientes categorías:

- Combinación colaborativa: Los algoritmos intercambian información, pero ninguno

es parte del otro. Ambos procedimientos se pueden ejecutar de manera secuencial, interconectados o en paralelo:

- Procesamiento secuencial. Uno de los métodos se ejecuta, y cuando éste finaliza, comienza el siguente algoritmo; de tal forma, que a veces podría parecer que el primero de ellos inicializa el segundo, o es el segundo módulo el que lleva a cabo un postprocesamiento de las soluciones obtenidas en el primer componente.

  En [60] se resuelve el problema de planificación *Production-line Scheduling Problem* (PLSP). Primeramente se emplea la heurística *Squeaky Wheel Optimization* (SWO) [126], llamada en repetidas ocasiones de forma aleatoria, para obtener soluciones factibles del problema. SWO utiliza un algoritmo voraz para construir los resultados. Seguidamente, y con las planificaciones obtenidas en la primera etapa, se ejecuta un método exacto mediante la herramienta MINTO (*Mixed INTeger Optimizer*) [181], que devuelve una solución, al menos tan buena como la mejor obtenida por SWO.

  En [134] se resuelve el problema del árbol de Steiner mediante una combinación de un algoritmo memético con un modelo entero, donde éste último realiza labores de postprocesado de las soluciones obtenidas por la primera técnica.

  En [190] se combinan los métodos de punto interior y metaheurísticas para resolver el problema de la mochila con múltiples restricciones (*Multiconstrained Knapsack Problem* -MCKP-). La primera fase consiste en la utilización del método del punto interior. Aplicando seguidamente un conjunto de heurísticos, se obtiene una agrupación de diferentes soluciones candidatas. Éste conjunto formará parte de la población inicial del algoritmo *Path Relinking*.

  A veces, una solución relajada del problema planteado puede ser un prometedor punto de partida para el método heurístico. Esta estrategia se utiliza en [200], donde mediante una LR, se obtiene un conjunto de soluciones que no son factibles para el problema original. Pero, aplicando un procesado sobre este conjunto, se genera la población inicial de entrada al algoritmo genético (GA).

  El trabajo [217] aborda el problema de planificación de trabajos *Job-Shop Scheduling Problem* comenzando desde una formulación ILP. Para cada variable, se toma el rango de los posibles valores, dividiéndose éstos en un conjunto de subrangos, que son indexados para poder ser utilizados con posterioridad. Los cromosomas del GA se definen de tal manera que cada posición representa una variable; y su valor se corresponde con el índice de uno de los subrangos de dicha variable. La aptitud de un cromosoma se calcula usando la relajación de Lagrange, para obtener un límite en la solución óptima sujeto a que los valores de las variables caigan dentro de los rangos correctos. Cuando finaliza el GA, se lleva a cabo una búsqueda exhaustiva de la región identificada como la más prometedora para obtener la solución final.

- Procesamiento paralelo o interconectado. Este tipo de combinación es menos frecuente, pero existen algunos ejemplos en los que los procedimientos exactos y heurísticos siguen este mecanismo de ejecución.

  En [215] y [216] aparece un *framework* llamado *asynchronous teams* (*A-Teams*),

que consiste en una arquitectura formada por un conjunto de agentes y memoria, conectados en una red de interconexión. Cada uno de estos agentes contiene un algoritmo de optimización capaz de tratar con un problema, con una relajación de éste, o con una subclase, también de éste. De esta forma, en esta arquitectura los agentes se encontrarán trabajando de forma asíncrona y autonoma, añadiendo, modificando o borrando soluciones que se encuentren en la memoria compartida. En [53] y [216] se ha empleado de forma satisfactoria para la resolución de algunos problemas de optimización.

- Combinación integradora: Una de las técnicas se encuentra incluida o embebida en la otra. De esta manera, el primero de ellas tendría un rol de componente "esclavo"; y la segunda, de "maestro".

  - Incorporación de métodos exactos en metaheurísticos.

    Primeramente se van a comentar el caso de los metaheurísticos que incluyen algoritmos exactos.

    Solución exacta de problemas relajados

    En el punto dedicado al procesamiento secuencial de métodos, se mencionó la utilidad de resolver la relajación del problema original. De esta manera, se pueden obtener soluciones iniciales prometedoras que pueden ser utilizados por otros métodos. Por ejemplo, en [58] y [199] se puede aplicar esta estructura a la búsqueda de vecinos, así como a las fases de recombinación, mutación y mejora local en algoritmos genéticos híbridos, con el objetivo de resolver el problema de la mochila.

    Aplicación de algoritmos exactos en la búsqueda local

    En [2] se presenta la técnica *Very Large-Scale Neighborhood* (*VLSN*), en la cual se emplean métodos exactos en procedimientos de búsqueda local a gran escala. En [43] se presenta un algoritmo exacto incluido en la fase de búsqueda local del procedimiento llamado *HyperOpt*. Este algoritmo lleva a cabo búsquedas exhaustivas de regiones prometedoras que son relativamente grandes en el espacio de soluciones. En [219] y [220] aparecen métodos exactos y heurísticos que participan en la resolución de un problema de partición. Y en el trabajo [197] se describe una combinación de algoritmos genéticos y métodos de ramificación y poda GA/B&B para resolver el problema del corte de vidrio (*Real-World Glass Cutting Problem*). El algoritmo *Branch & Bound* toma el resultado obtenido por el algoritmo genético, generando óptimos locales y mejorando, por tanto, la calidad de la solución. Por último, en [134], que ya se mencionó en el contexto de combinaciones secuenciales colaborativas, aparece otro tipo de composición exacto/heurístico: se utiliza un método exacto como una subrutina con la tarea de mejorar de forma local las soluciones candidatas del problema del árbol de Steiner.

Mezcla de soluciones

En [67] se presenta un espacio de trabajo (*framework*) en el que se combinan métodos de ramificación y poda y algoritmos evolutivos. En este caso se desarrolla el concepto de recombinación óptima, de tal forma que en el algoritmo evolutivo el operador explora el potencial de las combinaciones de los ancestros, utilizando un procedimiento de *Branch & Bound*, y así obtiene la mejor solución sin necesidad de ejecutar una fase adicional de mutación.

Con el propósito de resolver el problema de la coloración de grafos, el trabajo [161] presenta una algoritmo genético que incluye un método exacto especializado en dar solución al problema de asignación lineal *Linear Assignment Problem* (LAP). LAP se incluye en la fase de mezcla (*crossover*) del GA, generando una permutación óptima de colores. Como principal conclusión de este estudio, se puede afirmar que los procedimientos GA que contienen el algoritmo LAP en la etapa de mezcla, obtienen mejores resultados que los que no contienen dicho método exacto.

Algoritmos exactos como descodificadores

En los algoritmos evolutivos, a veces las soluciones candidatas se encuentran representadas de forma parcial o incompleta en el cromosoma. En estos casos, se puede utilizar un método exacto para determinar las partes omitidas del resultado obtenido previamente. En [210] se presenta un algoritmo híbrido que resuelve el problema de planificación en máquinas paralelas, minimizando el inventario y la acumulación de pedidos. Las soluciones se representan como un conjunto de productos asignados a cada máquina en un periodo. El tamaño óptimo de los lotes de productos se determinan mediante una formulación MIP (*Mixed Integer Linear Programming*).

- Incorporación de metaheurísticos en métodos exactos.

  A continuación se tratan técnicas exactas que incluyen metaheurísticos.

  Metaheurísticas para la obtención de soluciones y valores límite

  En [229] se examina una arquitectura que incorpora metaheurísticas en procesos de ramificación y poda. Una cuestión importante que se plantea es decidir si se debe iniciar una búsqueda desde un nodo del árbol B&B, de tal forma que, para tomar esta decisión, se proponen métodos metaheurísticos

  Metaheurísticas para la generación de columnas y cortes

  En los métodos *Branch and Cut* (B&C) y *Branch and Price* (B&P), se pueden incluir metaheurísticos que realizan tareas de separación dinámica de planos de corte y de generación de columnas respectivamente, con el objetivo de mejorar el rendimiento total de todo el proceso. El artculo [90] describe un enfoque de generación de columnas mediante heurísticos para resolver el problema del coloreado del grafo. Mientras que en el módulo maestro se emplea la herramienta *software* de resolución de problemas CPLEX, en la subrutina se utiliza algoritmos GA tanto para generar las columnas iniciales, como para encontrar las columnas con coste negativo que serán destinadas a la fase de generación de columnas.

En [194] y [195] se proponen nuevas formulaciones de programación entera ILP para el problema de la mochila de dos dimensiones, pero basado en tres etapas. A partir del modelo ILP, se desarrolla un algoritmo *Branch and Price* en el cual se lleva a cabo la tarea de generación de columnas mediante una jerarquía de procedimientos:

1. Un algoritmo voraz.
2. Un método evolutivo.
3. Resolver el *Pricing Problem* en su forma restringida mediante CPLEX.
4. Resolver el *Pricing Problem* en su forma completa mediante CPLEX.

Los experimentos llevados a cabo sobre ejemplos estándar de referencia *(standard benchmarks)* confirman los beneficios del nuevo enfoque. La combinación de los cuatro algoritmos en el marco propuesto de *Branch and Price* proporciona los mejores resultados en términos del valor objetivo medio, el tiempo medio de ejecución y el número de instancias resueltas de forma óptima.

Metaheurísticas para la guía estratégica de la búsqueda exacta

En [95] se presenta un procedimiento híbrido GA/B&B con el objetivo de tratar la viabilidad y la optimización de problemas de programación entera ILP. Primeramente comienza recorriendo el árbol B&B. Durante esta fase, la información de los nodos se recopila con el fin de sugerir cromosomas para ser añadidas a la aleatoria población inicial del algoritmo GA. Cuando se cumple un cierto criterio, el GA se inicia utilizando la población inicial aumentada por la información suministrada por los nodos B&B. Cuando el GA termina, su solución más apta se suministra al árbol B&B. El control total, por tanto, se devuelve al motor B&B, que examina los nodos recién agregados. Los resultados reportados en instancias de MAX-SAT muestran que este enfoque híbrido produce mejores soluciones que el B&B o el GA por separado.

En [146] se describen estrategias mejoradas de selección de nodos en métodos B&B, resolviendo modelos de programación entera mixta (MIP) mediante el uso de programación genética (GP): como consecuencia de la ejecución del algoritmo B&B durante una cantidad determinada de tiempo, se extrae información del árbol B&B para ser utilizado como conjunto de entrenamiento del módulo GP; cuyo objetivo es encontrar una estrategia de selección de nodos más apropiado para el problema específico en cuestión. La segunda fase del método B&B aplica la nueva estrategia suministrada por GP. Los resultados obtenidos concluyen que este enfoque tiene potencial, pero necesita ser mejorado para poder competir con las estrategias actuales de selección de nodos.

3.4.4. Combinación de métodos exactos y metaheurísticos: categorias alto/bajo nivel y *relay*/trabajo en equipo

Esta organización, presentada en [214], muestra un conjunto de categorías de alto y bajo nivel, así como "*relay*" y "trabajo en equipo" cuya finalidad es llevar a cabo una clasificación de procedimientos híbridos. Aunque en el caso anterior el objetivo será la cooperación únicamente

entre metaheurísticas, dicha clasificación se adaptará en [127] a una combinación de métodos exactos y heurísticos.

Primeramente se mostrarán aspectos de diseño que conciernen al procedimiento híbrido, para seguir con el proceso de la implementación.

Diseño de la Cooperación entre Métodos de Optimización

La cooperación involucra procesos de diseño e implementación. El primero de ellos trata aspectos de la arquitectura, de la funcionalidad y del algoritmo cooperativo en sí. La implementación, en cambio, tiene en cuenta la plataforma *hardware*, el modelo de programación y el entorno de la aplicación, entre otros. El diseño de los algoritmos híbridos se puede clasificar de la siguiente manera (adaptación de [214]):

1. Bajo nivel / Alto nivel
   - *a*) Bajo nivel: Un simple método de optimización.
   - *b*) Alto nivel: En este caso, los algoritmos se consideran como un componente autónomo y autocontenido.
2. Relevo (*Relay*) / Trabajo en equipo
   - *a*) Relevo: Un conjunto de métodos se aplica secuencialmente uno tras otro, cada uno utilizando la salida del anterior como su entrada, actuando, por tanto como un *pipeline*.
   - *b*) Trabajo en equipo: Representa modelos cooperativos de optimización.

Dicha organización queda reflejada en la figura 3.12.

Figura 3.12: Hibridación en la fase de diseño.

A continuación se expondrá con más detalle las categorias presentadas en esta etapa de diseño.

LRH (*Low-level Relay Hybrid*). Esta clase hace referencia a los algoritmos que tienen métodos embebidos en los procedimientos. La función incrustada se ejecuta de manera secuencial. Por ejemplo, podría ocurrir que una función global fuese dependiente de los resultados obtenidos por el método embebido. En el contexto de cooperación entre metaheurísticas, se podría dar el caso de un algoritmo evolutivo que ejecuta una búsqueda local, con el objetivo de realizar labores de intensificación sobre la mejor solución encontrada por el primer procedimiento. En matheurísticas, este tipo de cooperación se suele emplear cuando un heurístico se utiliza para mejorar un método exacto. Por ejemplo, la heurística podría estar diseñada para ayudar a definir la estrategia de búsqueda en el método exacto (selección del nodo a explorar, o generación de la columna, etc.) (ver figura 3.13). Un ejemplo de este tipo de cooperación se propone en [16]. En este estudio, se presenta un método B&C para resolver el problema de enrutamiento de vehículos con capacidad (CVRP). Se proponen diferentes enfoques heurísticos (heurística constructiva, algoritmos voraces y de búsqueda tabú) para optimizar la estrategia de búsqueda.

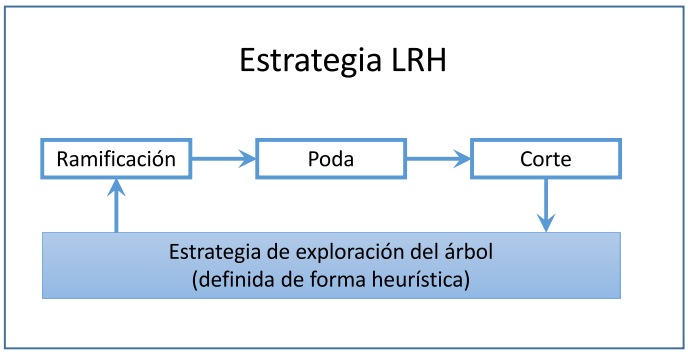

Figura 3.13: Cooperación LRH.

LTH (*Low-level Teamwork Hybrid*). En este caso, un método que se encuentra en un procedimiento puede ser reemplazado por otro, pudiendo mejorar, en algunas ocasiones, de forma radical una metaheurística. Pero a diferencia de la cooperación LRH, el método embebido se puede ejecutar en paralelo con el procedimiento global. En el contexto de la cooperación entre metaheurísticas, un buen ejemplo de este tipo de interacción entre métodos podría ser el algoritmo memético. Éste consta de un algoritmo genético (GA), cuyo operador de mutación es reemplazado por una búsqueda local. Ésta se aplica sobre los individuos que previamente han sido seleccionados de la población GA. Como se puede apreciar en la figura 3.14, se ejecuta una búsqueda exacta sobre soluciones parciales obtenidas por operadores heurísticos del algoritmo evolutivo, con el objetivo de intensificar la búsqueda, y encontrar óptimos locales de las nuevas soluciones.

Como se mencionó en la sección 3.4.3 del documento, y más exactamente en el apartado relativo a la "Incorporación de métodos exactos en metaheurísticos", en [67] se propuso un *framework* en que colaboran algoritmos genéticos y B&B; donde éste último ejercía de rol de operador dentro del GA. Tal y como se muestra en 3.15, éste es un ejemplo claro de una estructura colaborativa LTH.

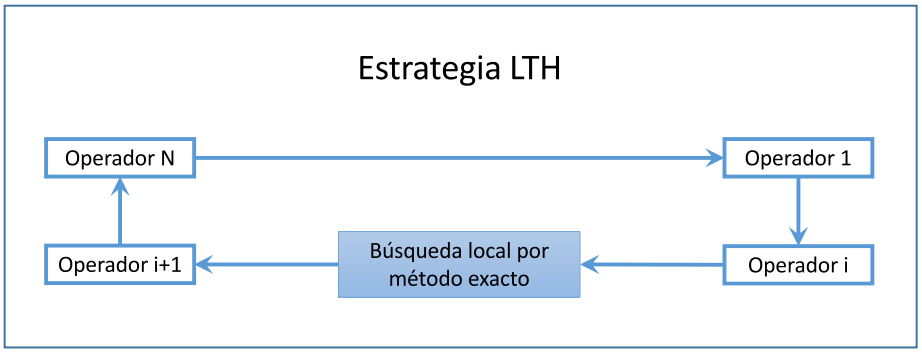

Figura 3.14: Cooperación LTH.

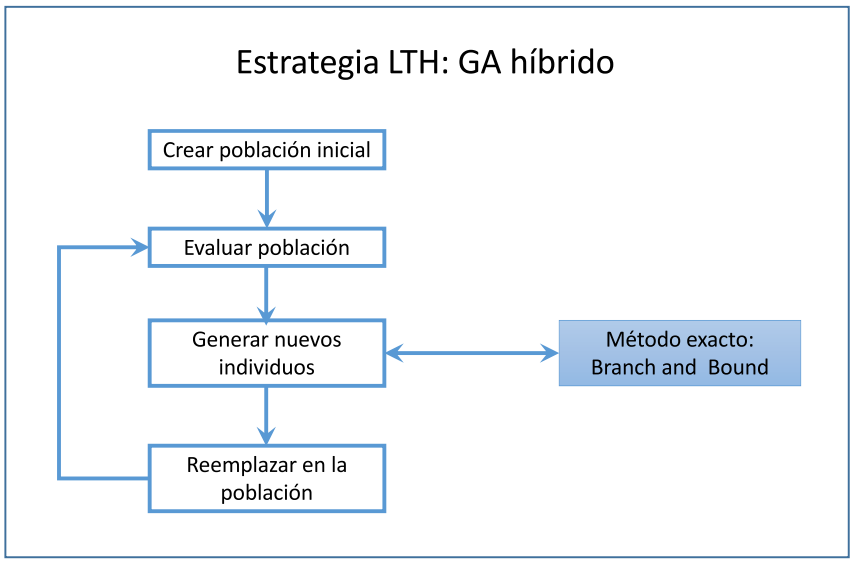

Figura 3.15: Algoritmo Genético Híbrido LTH.

Asímismo, se presenta otra organización LTH en la misma sección que se mencionó en el párrafo anterior (sección 3.4.3), pero en este caso, en el apartado de "Incorporación de metaheurísticos en métodos exactos". Se trata de una arquitectura híbrida [146], en la que un módulo de programación genética (GP), se encuentra incrustado en un proceso B&B, que a su vez se basa en un método entero mixto (MIP).

En [123] y [124] se proponen hibridaciones entre algoritmos genéticos y métodos exactos aplicados al problema del viajante *Travelling Salesman Problem* (TSP). En este caso, se reemplaza la operación de cruzamiento por un proceso B&B y un algoritmo que soluciona el problema *Spanning Tree* mínimo. Éste último (método) también proporciona las soluciones iniciales del GA.

Los algoritmos de búsqueda local a gran escala se adaptan a la estructura LTH. Estos procesos pueden ser descritos como procedimientos de búsqueda local en las que se hace uso de un gran espacio de rastreo para mejorar la eficiencia de la exploración. Esta tarea se puede llevar a cabo mediante métodos exactos o heurísticos. Se puede encontrar una revisión de estos procedimientos en [2]. Varios estudios proponen métodos exactos para llevar a cabo tareas de búsqueda local con el fin de encontrar la mejor solución en el subespacio de búsqueda. Esta solución ha sido presentada en [31] para resolver el problema TSP asimétrico, o en [206] que hace lo propio con el problema de enrutamiento de vehículos *Vehicle Routing Problem* (VRP).

HRH (*High-level Relay Hybrid*). En este clase de matheurísticas, los diferentes procedimientos, que son autónomos y autocontenidos, se ejecutan secuencialmente. Este esquema de cooperación en el más común en ser desarrollado para una hibridación de métodos de optimización.

Como en los otros esquemas de cooperación, se pueden considerar diferentes soluciones aplicadas con esta organización. Un enfoque habitual es diseñar una ejecución secuencial de un metaheurístico y una aproximación exacta (ver figura 3.16). La metaheurística está diseñada para dar información al algoritmo exacto, por lo que la información recibida podría ser utlizada para configurar los límites iniciales, y de esta manera, ayudar al método exacto a acelerar la búsqueda.

Figura 3.16: Organización HRH.

En [138] se propone una colaboración entre un algoritmo α-B&B y la metaheurística *Conformational Space Annealing* (CSA), con el objetivo de predecir la estructura de la proteina. α-B&B es un procedimiento de optimización global basado en en método B&B [13]; y CSA es un método estocástico que emplea elementos tanto de recocido simulado (*Simulated Annealing*) como de los algoritmos genéticos (GA) [152].

En [27] se investigan varios enfoques cooperativos para el problema *Biobjective Permutation Flow-shop Scheduling Problem*. Se diseña un procedimiento híbrido bi-objetivo: por una parte, un algoritmo memético adaptativo bi-objetivo, y por otra, un método exacto también bi-objetivo.

HTH (*High-level Teamwork Hybrid*). Esta clase contiene algoritmos donde los métodos autónomos realizan las búsquedas de manera cooperativa y paralela. Durante su ejecución, los diferentes algoritmos intercambian información, que depende del tipo de componente (ya sea heurístico o exacto). Por ejemplo, en la figura 3.17 se puede apreciar cómo dos algoritmos son lanzados en paralelo. Intercambian información de tal forma que el metaheurístico proporciona información al método exacto con el objetivo de reducir el espacio de búsqueda; y el exacto suministra una solución que será mejorada en el metaheurístico mediante un proceso de intensificación.. La principal dificultad es establecer los parámetros de comunicación entre procesos (cuándo y cómo se realiza el intercambio, por ejemplo).

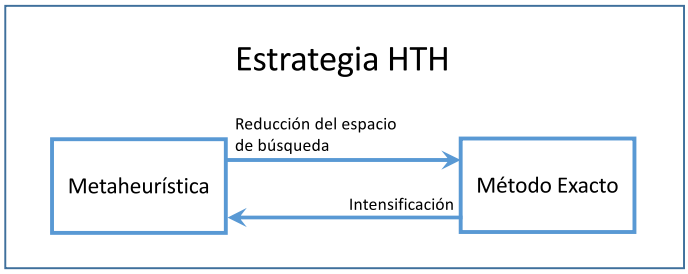

Figura 3.17: Organización HTH.

En [185] se combinan los métodos de recocido simulado y B&B, de tal forma que ambos procedimientos trabajan en el mismo problema, funcionando en paralelo, e intercambiando información cuando se cumplen unas determinadas condiciones. Este modelo se trata con más detalle en esta misma sección, pero en la parte dedicada al proceso de implementación.

Por último, [49] propone un modelo de cooperación entre una búsqueda local y un algoritmo *Column Generation* para resolver el problema VRP.

Otros Criterios de Diseño

En el diseño de matheurísticas se deben tener en cuenta otros aspectos relativos a la cooperación entre métodos:

- Resolución exacta o aproximada del problema
- Espacio de búsqueda parcial o global
- Naturaleza de la cooperación general o especializada.

A continuación, se comenta con más detalle cada una de estas consideraciones.

Resolución final exacta o aproximada del problema. La solución final del modelo cooperativo podría ser exacto o heurístico (aproximado). Los enfoques cooperativos exactos

toman información útil de la heurística para acelerar el rastreo de todo el espacio de búsqueda, actualizando los límites, encontrando soluciones iniciales o definiendo planos de corte útiles (ver modelo HRH, figura 3.16). Por otra parte, los modelos cooperativos heurísticos podrían utilizar los métodos exactos con el fin de obtener una solución inicial, que será mejorada mediante un proceso heurístico de intensificación.

Como se comentó anteriormente, en [49] se utiliza una búsqueda local para generar nuevas columnas con destino el algoritmo de ramificación y corte. De esta manera, el enfoque global de la solución es exacta, a pesar del uso de una metaheurística en el algoritmo cooperativo.

En [67] también se utiliza la misma perspectiva global. El metaheurístico trabaja sobre el mismo problema que el método exacto, proporcionando a éste último límites en el espacio de búsqueda para acelerar la resolución exacta.

En cambio, en [43] se describe una aproximación global heurística. En este estudio, el método exacto se incluye en un mecanismo de búsqueda local con el objetivo de explorar subespacios de búsqueda.

Cooperación de búsqueda parcial o global. Los componentes pueden actuar en todo el espacio de búsqueda o únicamente en una parte. Por un lado, en la cooperación de búsqueda global todos los algoritmos exploran el mismo espacio de búsqueda; y por otra parte, en la perspectiva de búsqueda parcial, el problema se descompone en sub-problemas, cada uno con su propio espacio de búsqueda.

La mayoría de los enfoques de cooperación metaheurística / exacta son a menudo parciales ya que el espacio de búsqueda es generalmente demasiado grande para el método exacto. De hecho, hay muchos ejemplos de cooperación parcial. A continuación se enumeran algunos de éstos:

- Un ejemplo es el método *Mimausa* diseñado por Mautor y Michelon en [171] para el problema de asignación cuadrática. El método construye en cada iteración un subproblema; y lo resuelve mediante un algoritmo de ramificación y cota (B&B).

- En [186] se propone una heurística para resolver el problema de la programación del proyecto con restricción de recursos (*Project Scheduling with Resource Constraint*). El modelo integra un método exacto en una heurística de búsqueda local. En cada iteración, el proceso genera un subproblema principal cuya resolución exacta proporciona un punto de partida para resolver el subproblema restante.

- Otros enfoques proponen resolver los subproblemas con un método exacto e integrarlos en el metaheurístico: en [41] para el problema de diseño de red de vuelo directo (*Direct Flight Network Design Problem*), o en [48] con el algoritmo *Forget-and-Extend Algorithm.*

- En [52], se propone un método híbrido que combina un algoritmo de búsqueda tabú y un método simplex, para hacer frente a la minimización global de funciones dependiendo de variables continuas. El algoritmo simplex se utiliza para acelerar la convergencia hacia un valor mínimo.

En cambio, en [193] aparece un modelo de búsqueda global. Proponen un enfoque exacto para resolver el problema *Flow Shop*. Una heurística específica calcula las soluciones para proporcionar los límites iniciales del algoritmo B&B. Los dos algoritmos exploran el mismo espacio de búsqueda, definiendo esta cooperación como global.

Naturaleza de la cooperación general o especializada. En un modelo híbrido de tipo general, todos los algoritmos resuelven el mismo problema de optimización, mientras que los componentes de una cooperación especializada resuelven diferentes problemas. Muchas de las soluciones propuestas pertenecen al tipo de cooperación general, como el presentado en [66]. En cambio, en [49] se muestra un ejemplo de cooperación especializada con el objetivo de resolver el problema de enrutado de vehículos con ventana de tiempo (*Vehicle Routing Problem with Time Windows* -VRPTW-). Uno de los métodos resuelve el problema del conjunto de cobertura (Set Covering Problem), y el otro el problema de enrutamiento. Este último tipo de cooperación se suele emplear cuando algunos de los problemas a resolver tienen algunas características particulares, de tal forma que se pueda solucionar mediante un modelo exacto.

Implementación

También se puede clasificar los algoritmos cooperativos desde el punto de vista de la implementación. Más exactamente se puede seguir dos enfoques: uno secuencial y otro paralelo.

Planteamiento secuencial de la implementación. La mayoría de las implementaciones propuestas son secuenciales. En muchos modelos se sugiere paralelizar con el fin de diseñar mejores esquemas de cooperación y acelerar el rendimiento y los tiempos de ejecución. La ejecución secuencial normalmente requiere una gran cantidad de recursos computacionales.

Implementación paralela. El despliegue del modelo en el *hardware* se puede llevar a cabo según la siguiente clasificación arquitectónica [6], [213] :

- Multiprocesador. Computador que cuenta con dos o más microprocesadores (CPUs); por lo que es capaz de ejecutar simultáneamente varios hilos pertenecientes a un mismo proceso o bien a procesos diferentes.
- Clúster. Conjunto o conglomerado de sistemas autónomos, unidos entre sí normalmente por una red privada de alta velocidad, y se comportan como si fuesen un único recurso computacional unificado.
- *Grid*. La computación *grid*, o de malla, es una tecnología que permite utilizar de forma coordinada recursos heterogéneos que no están sujetos a un control centralizado. En este sentido es una forma de computación distribuida, en la cual los nodos participantes pueden ser de iguales o distintas arquitecturas.

  Dentro de este último punto se podrían incluir la tecnología P2P, en la que se comparte la carga de trabajo entre pares, los cuáles ponen sus propios recursos informáticos a disposición de otros compañeros de la red, sin coordinación centralizada. O la computación

en la nube (*cloud computing*), que se basa en proporcionar recursos computacionales en demanda mediante Internet u otra red de alta velocidad.

Un punto de partida muy importante es determinar qué tipo de información se intercambia y en qué momento. En base a estas consideraciones, se pueden encontrar tres modelos de paralelización:

- Modelo paralelo a nivel de solución. El enfoque se centra en la generación en paralelo de cada solución. Las operaciones dependientes del problema aplicadas a las soluciones se realizan simultáneamente.
- Modelo paralelo a nivel de iteración. Este caso se trata de un modelo de trabajo que no altera el comportamiento de la heurística. Aprovecha los bucles en paralelo para la obtención de las soluciones.
- Modelo paralelo a nivel algorítmico. Varios procesos se lanzan simultáneamente para calcular las mejores soluciones. Éstos procedimientos pueden ser homogéneos o heterogéneos, independientes o cooperativos, partir de soluciones iguales o diferentes, y configuradas con parámetros iguales o diferentes.

En [66] se presenta una solución paralela para un enfoque híbrido GA/B&B. En este caso, los algoritmos GA se ejecutan de forma paralela, y un procesador dedicado ejecuta el método B&B sobre problemas restringidos. Este procesador suministra nuevos individuos a los GAs.

En [185], se hibridan los algoritmos *Simulated Annealing* (SA) y *Branch and Bound* (B&B) de forma paralela. Este modelo utiliza dos procesadores; en el primero de ellos, se ejecuta el recocido simulado; y en el segundo el algoritmo de ramificación y acotación. Entre ambos procesadores, se intercambian resultados de los límites calculados, así como información de la elección de variables. Los límites superiores obtenidos por el SA se pasan inmediatamente al código B&B. Si estos valores son mejores que los límites de B&B, éste actualiza dicho valor. Además, cualquier valor límite obtenido mediante la ejecución del B&B se transmite al método SA para que pueda ser utilizado. En segundo lugar, se intercambian información sobre la elección de variables. Una vez que tanto SA como B&B han calculado su respectivas clasificaciones de variables según la información de pseudocostes, se elabora otra lista de variables promediando las posiciones obtenidas en ambas estrategias.

En [6] se comenta la aparición de diferentes modelos cooperativos en redes P2P mediante la paralelización de metaheurísticas multiobjetivas híbridas [176], algoritmos evolutivos [149] [150] y combinaciones de GA y B&B [30].

Y en [7] se presenta la matheurística paralela H-DBFC, que resuelve problemas estocásticos mixtos 0 - 1 de gran tamaño.

Existen cada vez más *frameworks* que facilitan la paralelización en la hibridación de métodos. A continuación se presentan algunos de ellos:

- *A-Team* es una red de computación cíclica [215]. Consiste en una red de agentes que cooperan trabajando cada uno de ellos en los resultados de los demás. Cada agente es completamente autónomo (decide en qué resultados va a trabajar y cuándo).

- TECHS (*TEams for Cooperative Heterogeneous Search*) [72] permite el uso de agentes heterogéneos dentro de un equipo de búsqueda. Un agente puede ser tanto un método metaheurístico como exacto.
- En MALLBA (Universidad de MÁlaga, Universidad de La Laguna, Universidad de BArcelona), cada método de optimización está encapsulado en un esqueleto [5]. Se proponen diferentes métodos exactos (ramificación y acotación, divide y vencerás, programación dinámica); así como algoritmos heurísticos.

3.4.5. Conclusiones

La principal motivación que aparece detrás de la hibridación de diferentes algoritmos, es la de explotar el carácter complementario de las diversas estrategias de optimización, aprovechándose de la sinergia que puede existir entre ellas [36]. De hecho, eligiendo una adecuada combinación de los métodos, se pueden obtener buenos rendimientos en la resolución de problemas complejos de optimización. Para ello, hay que elegir tanto los métodos adecuados, como decidir la cooperación entre ellos. Desafortunadamente, desarrollar una solución híbrida de manera efectiva es una tarea ardua que requiere experiencia en diferentes áreas de optimización: en este caso, tanto de métodos exactos, como de procedimientos heurísticos. Además, es realmente complejo poder generalizar un modelo desarrollado, debido principalmente a que un cierto algoritmo híbrido podría funcionar bien para un tipo de problemas; pero en cambio, podría suministrar resultados discretos con otros.

Capítulo 4

Planteamiento del problema

4.1. Introducción

Después de realizar una revisión de los aspectos más importantes de los problemas de optimización, la complejidad de los algoritmos que resuelven éstos (y otros problemas), y los métodos de resolución más comunes, se presenta el momento de tratar el problema en cuestión que hay que resolver, tal y como se comentó en la introducción de la memoria. Y éste es el problema de la maximización de los caminos disjuntos (MEDP o EDP). En las siguientes líneas, se realizará una descripción más detallada del problema y se mencionarán algunos estudios de complejidad encontrados en el estado del arte. También se comentarán algunas implementaciones llevadas a cabo para resolver el problema, y que servirán de referencia para la comparación con la propuesta presentada en esta tesis. Asímismo, también se incluyen aplicaciones relacionadas con este problema de optimización.

Por otra parte, también en este capítulo se describirá en detalle el problema a resolver, incluyendo tanto las hipótesis que encaminarán los procesos de investigación, como los objetivos principales del problema.

4.2. El problema de los caminos disjuntos

El problema de los caminos con arcos disjuntos (*Edge-Disjoint Paths problem*, o EDP) se puede definir de la forma siguiente [24] [143]:

Dado un grafo $\mathcal{G} = (\mathcal{V}, \mathcal{A})$, y un conjunto $\mathcal{T}$ de solicitudes de conexión entre pares de nodos terminales $\mathcal{T} = \{(s_1, t_1), (s_2, t_2), ..., (s_k, t_k)\}$ en $\mathcal{V}$, se trata de determinar el mayor número de pares de nodos terminales (s_i, t_i), para todo $i = 1, 2, ..., k$ que se puedan conectar mediante caminos formados por arcos disjuntos.

También se puede definir de la siguiente manera:

Si $\tau = \{(s_1, t_1), ..., (s_m, t_m)\} \subset \mathcal{T}$ un subconjunto de pares terminales (origen, destino) de vértices en $\mathcal{G}$, se dice que τ es "realizable" en $\mathcal{G}$ si se cumple que los enlaces de las rutas $\{s_i, t_i\}$ para $i = 1, ..., k$ son disjuntos. De esta forma, el objetivo del ejercicio es encontrar el máximo conjunto realizable de τ.

La figura 4.1 muestra una instancia del problema, con un grafo no dirigido que contiene 12 nodos y 15 arcos. Para este ejemplo, hay 6 nodos terminales, que definen 3 pares terminales, siendo el conjunto $\mathcal{T} = \{(s_1, t_1), (s_2, t_2), (s_3, t_3)\}$.

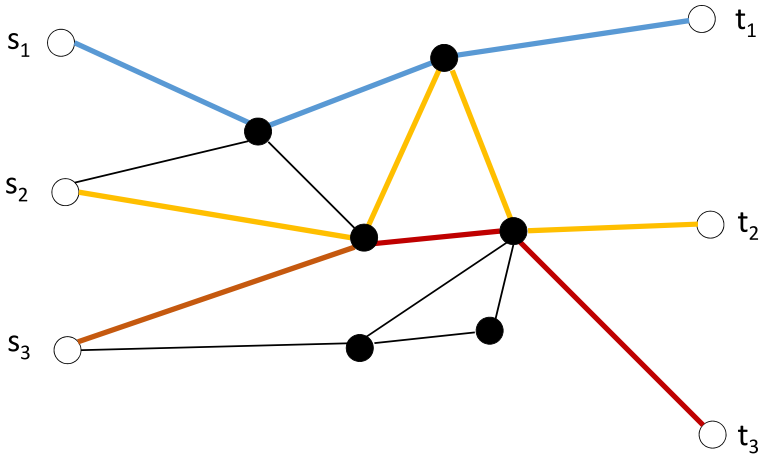

Figura 4.1: Solución del problema EDP para una instancia.

La resolución de este problema consiste en encontrar caminos entre pares terminales de tal forma que no se comparta ningún arco (aunque sí se pueden compartir nodos). En la anterior figura 4.1 se muestra una solución al problema, donde se han representado con diferentes colores (amarillo, azul y rojo) los arcos que pertenecen a cada uno de los tres caminos. Obsérvese que no es necesario ni pasar por todos los nodos del grafo, ni utilizar todos los arcos.

Por tanto, este ejercicio trata de asignar todos los pares terminales posibles, de tal forma que cada camino obtenido para cada par terminal no interfiera en los otros caminos. Este problema es también conocido como MDPP (*Maximum Disjoint Paths Problem*).

4.2.1. Problemas relacionados

Este problema esta relacionado con otros ejercicios parecidos. Sería similar al *Directed Edge-Disjoint Paths problem*, o abreviadamente problema DEDP [163]; pero en este caso, el grafo es dirigido. Otro ejemplo es el ejercicio de establecer caminos con nodos disjuntos (*Vertex-Disjoint Connecting Paths problem*, -VDP-).

El problema EDP presenta una formulación similar al problema de Steiner generalizado (*Generalized Steiner Problem* -GSP-) [182], que permite modelar el diseño de redes de comunicaciones de alta fiabilidad. En el GSP aparecen los mismos elementos que en el EDP y, adicionalmente, aparece una matriz de costes asociada a las aristas del grafo, y otra dedicada a informar sobre los requerimientos de conectividad asignados a cada par terminal (cantidad de caminos disjuntos en aristas requeridos para cada par de vértices terminales). Se trata de encontrar un subgrafo de coste mínimo que cumpla con los requisitos de conexión definidos, validándose para cada par terminal de forma independiente.

Otra variante de estos problemas consiste en considerar un grado de congestión acotado por los arcos (problema EDPwC o *Edge-Disjoint Paths with Congestion)* [12]. En este tipo de problemas, también el objetivo es establecer el mayor número de rutas disjuntas entre

pares terminales; pero con la restricción de que un máximo de $c_{u,v}$ (parámetro de congestión o capacidad del enlace) caminos pueden circular a través de cada arco (u, v). Una generalización de este último es el *Multi-commodity Flow Problem*. En este caso, además de capacidad de arcos, se debe satisfacer la demanda d_i de cada par terminal i. El objetivo es, por tanto, maximizar la suma de las demandas que pueden ser enrutadas completamente, cumpliendo con las restricciones de capacidad de los enlaces. Si se permitiesen flujos fraccionales, compartidos en diversas rutas para cada par terminal, el problema sería P, y se podría resolver mediante un programa lineal [64]. En otro caso, si el flujo es entero, el problema es NPC [89]. Cuando las demandas y capacidades son igual a 1, se obtiene el problema MEDP generalizado [135], [141].

Routing and Wavelength Assignment Problem (RWA) es un problema aplicado a redes óptimas. Pero en este caso, el objetivo es minimizar el número de arcos que comparten rutas; porque como consecuencia, el número de longitudes de onda (*wavelengths*) a utilizar en el total de conexiones también será menor [9]. Éste es un problema NPC, debido a que se puede reducir [56] al problema de coloración de grafos (*Chromatic Number Problem*), uno de los 21 problemas NPC originales de Karp. También se puede reducir al problema MPC *Multi-commodity Flow Problem* [89].

Para finalizar, hay que destacar que EDP es diferente al problema que maximiza el número de caminos con enlaces disjuntos, que conectan un único par terminal. Este ejercicio se conoce en la literatura como *Edge-Disjoing Menger problem* (EDM) [34]. EDM se puede resolver en tiempo polinómico [3], a diferencia del EDP, que es NP.

4.2.2. Aplicaciones

En las redes de comunicaciones, un nodo puede ser un equipo de transmisión de información (módem, concentrador, puente o conmutador) o un equipo terminal de datos (teléfono, impresora o servidor -*host*-). Una red suele ser modelada como un grafo no dirigido $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, donde el grupo de nodos $\mathcal{V}$ corresponde a equipos terminales / de comunicaciones, y el conjunto de aristas $\mathcal{E}$ incumbe a los enlaces físicos. Cada enlace $e \in \mathcal{E}$, con $e = (i, j)$ donde $i, j \in \mathcal{V}$, tiene un peso $w(e) \in \mathcal{R}^+$, que normalmente representa la distancia física entre ambos extremos. Sea $\mathcal{T} = \{(i_k, j_k) | i_k \neq j_k \in \mathcal{V}, i \leq k \leq K\}$ un conjunto de K pares de nodos, llamados "nodos terminales", que solicitan ser conectados con una ruta (o camino) en $\mathcal{G}$. La definición de caminos disjuntos puede referirse a nodos, enlaces o ambos. En este caso se centrará en la variante más estudiada, que es aquella en la que implica no compartir enlaces. Por tanto, el problema de maximización de camino con arcos disjuntos EDP consiste en calcular el mayor número de rutas con enlaces disjuntos que conectan los nodos terminales en $\mathcal{T}$.

Existe una amplia variedad de problemas de red en los que se producen varias peticiones de conexión de forma simultánea [143]. En general, se atiende cada solicitud buscando y encontrando una ruta en la red, donde el origen y destino de dicho circuito son aquellos servidores o *hosts* que desean establecer un camino para el envío de datos a través de ellas. El intercambio de información mediante rutas disjuntas incrementan el ancho de banda efectivo, la velocidad y la probabilidad de recibir los datos correspondientes. También se reduce la congestión y posibles fallos en la comunicación [25], [28]. En el enrutamiento de circuitos virtuales [20], [191], [119], [10] se necesita reservar previamente un camino para cada petición

de transmisión, de tal forma que una vez establecida la comunicación, no se producirán interrupciones. En [10] se presenta su uso en servidores de bases de datos y de vídeo.

Este problema también se puede utilizar en el diseño VLSI [131], [153], [50]. Es este caso, se requiere maximizar el número de rutas de cableado disjuntos en la malla o *grid* entre los distintos pares de terminales, evitando el solapamiento.

4.2.3. Complejidad del problema

Decidir cómo se conectan cada uno de los pares terminales es un ejercicio NP-completo [99]. En este libro, que se ocupa de más de 300 problemas NP-completos, se encuentra la cuestión ND40 de conexión de rutas disjuntas (*Disjoint Connecting Paths*). Obtiene dicha complejidad mediante su transformación desde el problema 3-SAT [139], [129] y [158] , que es uno de los 21 problemas clásicos NPC de Karp [130].

Este problema, básicamente, indica que teniendo un grafo $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, con $\mathcal{V}$ los vértices y $\mathcal{E}$ los enlaces de la estructura y una colección de pares de nodos disjuntos $(s_1, t_1), (s_2, t_2), ..., (s_k, t_k)$, se pregunta si $\mathcal{G}$ contiene k rutas con vértices disjuntos, cada cual conectando s_i a t_i, para cada i, $1 \leq i \leq k$. Este ejercicio es semejante al problema de maximización de caminos disjuntos (EDP) tal y como aparece en la recopilación de problemas NP en [68]. Asímismo, en este listado se indica que el problema EDP es también similar a los problemas *Minimum Unsplittable Flow* y *Minimum Path Coloring*.

Por otra parte, y tal y como se comenta en [34], es también NP-completo para tipos específicos de grafos como los grafos planos [180] y [225], serie-paralelos [184], y grid [170] y [147].

Se necesitaría una máquina no determinista de Turing para obtener en tiempo polinominal el resultado del problema. Asímismo, hay que añadir a la problemática anterior, la dificultad de verificar el resultado propuesto. Este proceso, al igual que el paso de resolución, tampoco se consigue realizar en tiempo polinomial mediante una máquina determinista.

Una explicación más detallada de la complejidad del problema puede ser obtenida mediante los "algoritmos de aproximación". Pero antes de continuar, hay que clarificar algunos detalles respecto a estos procedimientos:

- Un algoritmo de aproximación (*approximation algorithm*) aborda los problemas de optimización en tiempos polinómicos, y asegura unas soluciones de calidad acotadas por un ratio respecto a la solución óptima del problema [125].

- Los algoritmos metaheurísticos (*approximated algorithms*), comentados en el capítulo 3.3, son también métodos de aproximación que encuentran soluciones razonablemente buenas en tiempos normalmente rápidos. Pero en este caso, no se asegura una cota de calidad respecto a la solución óptima.

  En algunos problemas de optimización, las metaheurísticas son la única opción viable.

Con respecto al primero de ellos (*approximation algorithm*), se tienen los siguientes algoritmos [34]:

- Algoritmo de aproximación-ρ para un problema Π. Dado un ratio de aproximación $\rho > 1$ y una instancia x de Π, el algoritmo devuelve una solución de x con coste de casi ρ veces la solución óptima y tiempo polinómico.

  Varía ampliamente su aproximación en problemas NP-difícil. Algunos de ellos pueden ser aproximados mediante factores arbitrarios, mientras que en otros, esencialmente no se encuentran aproximación. Por esta razón, a veces las heurísticas o metaheurísticas son la única forma de poder abordar estos tipos de problemas.

  Los problemas que son aproximados con $\rho > 1$ pertenecen a la clase de complejidad APX.

- Aproximación en tiempo polinómico (PTAS- *polynomial time approximation scheme*-). Dado un valor ε tal que $0 < \varepsilon < 1$, PTAS, es un algoritmo de aproximación-ρ con $\rho = (1 + \varepsilon)$.

  Es una subclase de APX.

Obtener *approximation algorithms* es muy deseable porque se obtendrían soluciones cercanas, mediante un ratio, al óptimo. El inconveniente que se presenta es similar a los aportados para los métodos exactos. Por una lado, se necesita una gran base matemática, lo que incide en la dificultad del desarrollo. Y también presentan menor flexibilidad para adaptarse a las diferentes variantes del problema.

Un problema se dice que es APX-*hard* (APX-difícil) si existe alguna constante $\varepsilon > 0$ tal que obtener un algoritmo de aproximación-ρ con ratio $(1 + \varepsilon)$ es un problema NP-duro. Por tanto, será un ejercicio difícil de aproximar, debido a la complejidad de obtener el algoritmo de aproximación PTAS asociado, a menos que $P = NP$ [15].

Una vez comentados los diferentes algoritmos de aproximación, se puede señalar que para diversos tipos de grafos, el problema EDP pertenece a la clase de problemas APX-difíciles (APX-*hard*) [83], [100], [113] y [159]. Por otra parte, la aproximación del problema EDP se ha llevado a cabo heurísticamente mediante algoritmos voraces [34] [47], de colonia de hormigas [33], [34], o genéticos [120].

Este problema es de interés para la teoría de grafos. Y aunque es de complejidad NPC, se han realizado diversas investigaciones en identificar casos especiales que puedan ser resueltos en tiempo polinómico; y existen trabajos [94], [136], [137] y [135] que obtienen algoritmos de aproximación para este ejercicio. Se han conseguido procesos del orden de $O(log\, n)$ para grafos casi eulerianos, grafos planos con un diámetro uniformemente grande (que incluyen un nivel de red de dos dimensiones y subredes locales planos). Asímismo se han logrado crear procedimientos exactos para un número fijo de pares terminales, y para el caso de un grafo embebido en una superficie fija, tal y como se describen en los trabajos de Kleinberg y Tardos [137], Robertson y Seymour [202], y Frank [94].

Los grafos aleatorios $G_{m,n}$, para todas las densidades de enlaces, son tratados en [40], obteniéndose, en este caso, un tiempo polinomial en la búsqueda del número de rutas disjuntas óptimas. Se consigue un límite para este problema cuando se utilizan grafos no triviales. En grafos dirigidos se ha obtenidos resultados del orden de $O(\sqrt{\mid E \mid}$ en algoritmos de aproximación [135], [28], [142], [145] y [209]. Y se han alcanzado, de la misma manera que en los grafos

dirigidos, mejores ratios de aproximación con algunas subclases de grafos planos, completos, aleatorios o expandidos [226].

El uso y configuración online del algoritmo es importante en aplicaciones en los que las peticiones llegan a lo largo del tiempo, y deben ser procesados inmediatamente. Estos procedimientos online se han obtenido en [19]. En [137] se ha conseguido un algoritmo que tiende a $O(log\, n)$ para grafos casi eulerianos.

A continuación se van a comentar algunos estudios de complejidad para determinados casos del problema.

EDP es un problema NP-completo si el grafo es dirigido

En [133] y [59] muestran resultados de complejidad obtenidos por algoritmos de aproximación en la resolución del problema. Cuentan que el problema es NP-difícil para grafos dirigidos. Este análisis se realizó en [93] y [87], aplicando el problema de "homeomorfismo" de subgrafos dirigidos. A continuación se comenta el ejercicio mostrando previamente y de forma breve el concepto de homeomorfismo de grafos:

Sea $e = (v, w)$ un arco de un grafo G. La "subdivisión" de la arista e consiste en agregar un nodo u a G tal que $u \notin V$, y reemplazar la arista e por dos aristas $e' = (v, u)$ y $e'' = (u, w)$. Un grafo G' es una subdivisión de otro grafo G si G' se puede obtener de G por sucesivas operaciones de subdivisión.

Dos grafos G y G' son "homeomorfos" $G \cong_H G'$ si hay un isomorfismo entre una subdivisión de G y otra de G'. En teoría de grafos, un "isomorfismo" entre dos grafos G y H es una biyección f entre los conjuntos de sus vértices $f : V(G) \to V(H)$ que preserva la relación de adyacencia. Es decir, cualquier par de vértices u y v de G son adyacentes si y sólo si lo son sus imágenes, $f(u)$ y $f(v)$, en H. El problema de homeomorfismo de subgrafos dirigidos tratado en [93] intenta determinar si un grafo P es homeomórfico a un subgrafo G de entrada al problema. De esta forma, se relacionarían nodos de P con nodos de G, y arcos de P con caminos de G. Los grafos G y P pueden ser ambos dirigidos o no dirigidos. Y las rutas de P se corresponden a arcos de G a nivel de pares de nodos disjuntos.

Este problema puede ser visto como un ejercicio de búsqueda de rutas generalizado. Si por ejemplo, el grafo H consiste en dos arcos disjuntos, y se tiene la correspondencia de nodos, entonces el problema es equivalente a buscar un par de caminos disjuntos para los vértices especificados en el grafo de entrada G. Dado un par de grafos (H, G), y una correspondencia de nodos como entrada. Este problema es NPC si se cumplen los siguientes puntos:

- G contiene un subgrafo homeomórfico a H, y
- $P \neq NP$.

Además, como se ha visto en líneas anteriores, el problema de homeomorfismo entre grafos, incluye también un problema de isomorfismo entre grafos. Y éste último es un problema de decisión NP-completo que se demuestra mediante su reducción al problema de la *clique*. El teorema de Cook-Levin ya demostró que el isomorfismo del grafo era NP-completo, usando una reducción de $3 - SAT$ que involucraba al problema de la *clique* [62].

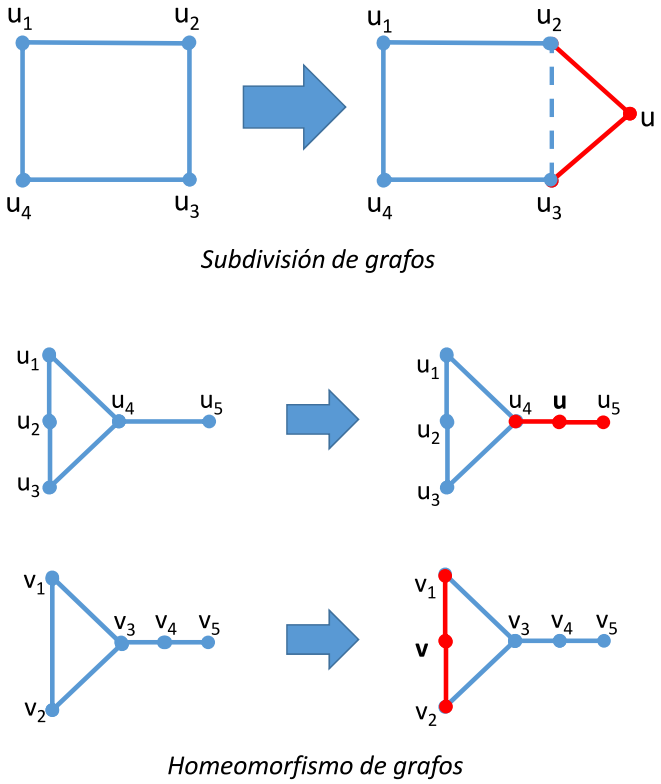

Figura 4.2: Subdivisión y homeomorfismo de grafos.

Tiempo polinomial para un número fijo de pares terminales

En [202] y [87] se describe un algoritmo que resuelve para un valor k fijo y tiempo polinomial, el siguiente problema:

Dado un grafo $G(V, A)$ y k pares de vértices terminales de G, se calcula si existen k caminos de nodos disjuntos que enlazan los pares terminales.

El tiempo de ejecución estaría del orden de $O(\mid V(G) \mid^3)$. Este problema es NP-completo cuando k es variable. Pero, para k fijo, el problema es más manejable. En [205], [207] y [218] se presentan algoritmos polinómicos para resolver el problema con $k = 2$. Pero para grafos dirigidos y $k = 2$ es NP-completo (ver sección anterior) [93].

Complejidad del problema en grafos planos y en otros tipos especiales de estructuras

En [204], se trata la complejidad del problema para el caso especial de los grafos planos.

En teoría de grafos, un grafo plano (o *planar* según referencias) es un grafo que puede ser dibujado en el plano sin que ninguna arista se cruce. Una definición más formal puede ser que este grafo pueda ser "incrustado" en un plano. En dicho artículo, también se continúa comentando que el problema de la "disjunción" en grafos (ya sea arcos o vértices disjuntos, y tanto para grafos dirigidos como no dirigidos), es un problema NP-completo, tal y como se comenta en [129].

Por otra parte, el problema se resuelve en tiempo polinómico cuando el número de pares terminales k es fijo y el grafo es no dirigido (ver apartado anterior) [202]. En el caso de los grafos dirigidos, el problema es NP-completo, incluso para $k = 2$ [93] y [87]. En similares circunstancias, con k fijo, [93] y [87] comentan que en grafos dirigidos acíclicos (sin ciclos) el problema de arcos disjuntos es también de resolución polinomial. Para el caso de los grafos planos y un valor k no fijo el ejercicio es NP-completo, tal y como se desarrolla en [147] para grafos de malla (*mesh*), y en [180] para grafos planos con un grado máximo de 3 [158]. Pero se puede resolver en tiempo polinomial cuando G es plano y los terminales se encuentran en un número limitado de caras [1] de G [94]; por ejemplo, si todos los pares terminales se encuentran en la región exterior.

Por último, en árboles no dirigidos, se ha encontrado un algoritmo polinomial en [100].

El algoritmo voraz que resuelve el problema EDP es un ρ-algoritmo de aproximación, con $\rho = \sqrt{m}$

Por otra parte, en [144] se desarrolló un algoritmo voraz para la resolución del problema. Este procedimiento tiene la particularidad de no tomar las peticiones de conexión de forma aleatoria. Es decir, en este caso, se acepta las peticiones en orden creciente de longitud; lo que implica que se acepta una solicitud T_i tal que no exista ninguna otra solicitud pendiente cuyo camino más corto sea menor que la correspondiente a T_i [86]. En [142] y [87] se indica que el algoritmo voraz simple (ver código 4.1) que resuelve el problema EDP es un ρ-algoritmo de aproximación, tanto para grafos dirigidos y no dirigidos, con $\rho = \sqrt{m}$, donde m es el número de enlaces. A continuación se muestra una pequeña demostración:

Suponiendo que se obtiene una solución óptima S^*, tal que $\mid S^* \mid = OPT$.

Sea $m = \mid E \mid$ el número de enlaces total del grafo $G(V, E)$.

Sea m_0 el número de enlaces utilizados por rutas de S^*.

Mientras que el algoritmo acepte rutas que son más cortas que $\sqrt{m_0}$, cada una de esas rutas intersecta con a lo sumo $\sqrt{m_0}$ rutas de S^*.

Cuando el proceso considera aceptar rutas con longitud de al menos $\sqrt{m_0}$, las rutas de S^* tienen longitud de al menos $\sqrt{m_0}$, por lo que habrá, a lo sumo, $\frac{m_0}{\sqrt{m_0}} = \sqrt{m_0}$ número de rutas en S^*.

Por lo anterior, el procedimiento contendrá al menos $\frac{OPT}{\sqrt{m_0}}$ rutas.

Entonces, el algoritmo voraz es un ρ-algoritmo de aproximación, con $\rho = \sqrt{m}$.

Asímismo, en el mismo estudio, [144], se indica que el algoritmo voraz obtiene un radio de aproximación $O(d^*)$ en instancias cuyas solución óptima tenga una longitud de camino medio de d^*.

[1] Las caras son las regiones del grafo conectadas por aristas, incluyendo la región externa e infinita.

Algoritmo 4.1: ρ-`Greedy Algorithm`

```
Input : T {set of shortest paths}
Output : T' {solution: edge-disjoint paths}
1: T' ← ∅
2: while T ≠ ∅ do
3:    {(s_i, t_i) is a pair from T, with the less number of arcs}
4:    (s_i, t_i) ← getShortestPath(T)
5:    T' ← addPath(s_i, t_i)
6:    T ← deletePath(s_i, t_i)
7: end while
8: return T'
```

El problema es NP-difícil para obtener un algoritmo de aproximación $m^{\frac{1}{2}-\varepsilon}$ con $\varepsilon > 0$ para grafos dirigidos

En [112] y [113] se estudian la obtención de un algoritmo de aproximación $m^{\frac{1}{2}-\varepsilon}$ en grafos dirigidos con $\varepsilon > 0$: se demuestra que es NP-difícil [87].

- El teorema que se pretende demostrar es el siguiente: dado

  un grafo $G(V, A)$, y

  un conjunto de pares terminales $T = \{(s_i, t_i) : i \in [k], s_i, t_i \in V\}$

  Entonces, para cualquier $\varepsilon > 0$, distinguir si k pares de T pueden ser conectados mediante caminos con arcos disjuntos, o a lo sumo una fracción $\frac{1}{m^{\frac{1}{2}-\varepsilon}}$ de los k pares que pueden ser conectados es NP-difícil.

- Demostración mediante reducción al problema $2DIRPATH$: dado

  un grafo dirigido $H(V, A)$, y

  un conjunto de vértices $\{(x_1, x_2, y_1, y_2\} \in V$

  ¿Existen dos caminos con arcos disjuntos que conecten x_1 con y_1 y x_2 con y_2, siendo este problema NP-completo?

  Con lo anterior, lo que se pretende es relacionar instancias del problema EDP con instancias del problema $2DIRPATH$:

  Si existen rutas de arcos disjuntos de x_1 a y_1 y de x_2 a y_2 en H, entonces existen N caminos de arcos disjuntos en G, que conecta s_1 con t_1 para cada $i \in N$.

  De esta forma las instancias de $2DIRPATH$ con las dos rutas satisfechas se relacionan con las instancias de EDP con los N caminos conectados

Algoritmo de aproximación en grafos casi eulerianos con factor constante

En grafos de malla, se obtuvo una aproximación del orden de $O(log\, n \cdot log(log\, n))$ para un algoritmo *online* [2] en [20]. También se obtuvo una aproximación $O(log\, n)$ en [17] y [136]. Pero en [137] se consigue un algoritmo de aproximación de orden $O(1)$ para mallas de dos dimensiones. Asímismo, para los grafos "casi"-eulerianos también se alcanza un ratio de $O(1)$. Y presenta un ratio de aproximación de $O(log\, n)$ en un algoritmo *online* para grafos de malla.

Se recuerda que un ciclo "euleriano" es aquel camino que recorre todas las aristas de un grafo tan sólo una única vez, siendo condición necesaria que regrese al vértice inicial de salida[3].

Algoritmo de aproximación en otros tipos de grafos

Tratando el mismo procedimiento voraz descrito en el algoritmo 4.1, en [51] se dedujo que el ratio anterior de $O\sqrt{n}$ no se ajustaba bien en los grafos densos[4]. En este caso, concluyeron que el ratio de aproximación era $O(min(\sqrt{m}, n^{\frac{2}{3}}))$ para grafos no dirigidos; y $O(min(\sqrt{m}, (n \cdot logn)^{\frac{2}{3}}))$ para grafos dirigidos. En el mismo artículo, logran un algoritmo combinatorio del orden $O(\sqrt{m} \cdot logn)$ para grafos dirigidos sin ciclos [87].

Siguiendo con casos especficos de grafos, en [84] se indica que para cualquier valor fijo de ε, tal que $\varepsilon > 0$, existe un algoritmo de aproximación con $\rho = \frac{5}{3} + \varepsilon$ para árboles bidireccionales.

Para finalizar, el problema es NP-duro para grafos completos[5] no dirigidos [85], sin haber encontrado ningún resultado de no aproximación.

4.2.4. Resoluciones previas del problema

En este punto se van a comentar diferentes propuestas encontradas en el estado del arte, y que servirán de referencia para realiza un análisis comparativo respecto a la solución planteada en este documento.

Algoritmo de colonia de hormigas

En [33] se presenta al algoritmo ACO que soluciona el problema EDP. Cada problema $\Pi = (G, T)$ del problema EDP puede ser descompuesto en $|T|$ subproblemas tal que $\Pi_j = (G, T_j)$, con $j \in \{1, ..., |T|\}$. Con el objetivo de buscar una ruta para el par terminal $T_j \in T$, se va a asignar una "hormiga" a cada uno de los subproblemas. La construcción de una solución consiste en cada hormiga j construyendo una ruta Π_j entre los dos puntos terminales de T_j. De forma obvia, ningún subproblema es independiente del conjunto de las $|T|$ rutas que se van construyendo, debido a la restricción de arcos disjuntos.

En [34] se actualiza el algoritmo anterior, incluyendo las siguientes características:

[2] Un algoritmo *online* no tiene toda la entrada disponible al comienzo del algoritmo. A diferencia del *offline*, que necesita disponer de toda la información de entrada antes de la ejecución

[3] Un ciclo es un camino en un grafo donde coinciden vértice inicial o de salida y vértice final o meta. Una definición más formal lo expresa como aquel ciclo que contiene todas las aristas de un grafo solamente una vez. Se debe tener en cuenta que no importa la repetición de vértices mientras no se repitan aristas. Un grafo euleriano contiene un ciclo euleriano.

[4] Un grafo denso, en matemáticas, es un grafo en el que el número de aristas está cercano a su número máximo. Lo opuesto, un grafo con solo algunas aristas, es un grafo disperso.

[5] En teoría de grafos, un grafo completo es un grafo simple donde cada par de vértices se encuentra conectado por una arista.)

- La construción de las soluciones se realizan de forma paralela, a diferencia del ACO básico que los generaba de forma secuencial.
- Utilización de una lista de candidatos para restringir las diferentes opciones a considerar en la generación de las soluciones. Se tomarán únicamente los mejores aspirantes.
- Se mejora la fase de actualización de las "feromonas" en la búsqueda de caminos.
- Cuando el algoritmo es incapaz de superar la mejor solución durante un número determinado de iteraciones, se ejecuta un mecanismo de "escape". Este proceso consiste en destruir de forma parcial una solución para intentar salir de los óptimos locales.

En el escrito se realiza una comparativa con un procedimiento MSGA. ACO obtiene mejores resultados, tanto en tiempo de proceso como en pares terminales conectados.

Constrained Optimum Path

Los problemas del árbol óptimo restringido (COT) y del camino óptimo restringido (COP) aparecen en diversas aplicaciones de la vida real, tales como redes de telecomunicaciones y transporte. Estos problemas consisten en encontrar uno o más árboles -COT- (o caminos -COP-) en un grafo dado que satisfagan algunas restricciones dadas, mientras minimizan o maximizan una función objetivo. En [80], se muestra una propuesta *Constrained Optimum Path* (COP) para resolver el problema EDP. Presenta al algoritmo LS-SGA cuya idea principal es llevar a cabo una búsqueda local con el objetivo de minimizar el número de restricciones infringidas. Para ello, se inicia con un conjunto de soluciones aleatorias que conectan todos los pares terminales. En la siguiente etapa, se realizan "movimientos" locales que minimicen el número de enlaces no disjuntos.

Se extraen de forma aleatoria rutas que conectan pares terminales de la solución. Y sobre el grafo que queda sin los enlaces recientemente separados de la solución, se utiliza un algoritmo voraz para buscar nuevas soluciones. Si se encuentran mejores resultados, se actualiza la mejor solución.

Partiendo de una solución inicial vacía, LS-R ("recursiva") emplea algoritmos voraces para crear resultados. Seguidamente se toman grafos en los cuales no se encuentran los arcos escogidos en las soluciones obtenidas por el algoritmo *greedy*; y de forma recursiva se vuelve a utilizar el procedimiento voraz para seguir creando nuevas soluciones.

Algoritmo genético

En [120] se muestra un algoritmo genético que lleva a cabo el proceso de resolución del problema. Específicamente, cada individuo de la población es codificado como un conjunto de $|T|$ caminos, donde cada trayecto es representado por una lista de valores reales con longitud $|V|$ en el intervalo $[0, 1]$. Cada uno de estos valores reales denota la "prioridad" de un nodo a ser seleccionado en el camino en construcción. De una forma más precisa, dado un par de terminales, $[i_k, j_k]$, y comenzando el camino en origen i_k, los nodos candidatos a ser incorporados al camino serán aquellos que no hayan sido previamente utilizados, y además contengan valores altos de prioridad. Entonces el método comprueba si la inclusión del nodo seleccionado produce

una solución factible (incluyendo por tanto el vértice a la ruta) o no (retrocediendo tantos pasos como sea necesario). El proceso de construcción del camino se detiene cuando el nodo seleccionado es j_k.

El operador de cruce (*crossover*) del GA genera un hijo mediante una combinación lineal de dos padres. Por otra parte, el operador de mutación invierte el valor real correspondiente (es decir, las prioridades altas se transforman en pequeñas y viceversa), introduciendo diversidad en las soluciones. GA también considera un operador de "autoadaptación" que mejora la calidad del individuo, en particular. De forma más específica, intenta conectar aquellos pares terminales no conectados en la solución, sustituyendo los valores reales con información de la ruta más corta, o la longitud del camino, entre otros criterios.

Técnica de paso de mensajes

Recientemente, en [10] proponen un método distribuido para resolver el problema basado en técnicas de paso de mensajes MP (o método de la cavidad -*Cavity Method*-) [21], [179], [178] y [177].

Los autores primeramente describen cómo EDP puede ser fácilmente resuelto en árboles. Entonces, presentan la ecuación recursiva de la suma mínima para ser utilizada en el proceso de paso de mensajes. En el caso de grafos localmente arbóreos, como los dispersos, el método obtiene soluciones aproximadas para el problema EDP. Con el fin de mejorar aún más el rendimiento, el procedimiento propuesto transforma la ecuación recursiva antes mencionada en el problema de emparejamiento de pesos máximos (*Maximum Weight Matching Problem)* [99]. Esta transformación permite al algoritmo ejecutar cada iteración del método en tiempo polinomial (en el número de aristas) y tiempo lineal (en promedio) para grafos dispersos. Además del algoritmo MP, y con el fin de ayudar y acelerar la convergencia de las ecuaciones MP, se utilizó una técnica de refuerzo (*reinforcement*) [39] y [11], obteniéndose el procedimiento MP_*reinf*. Por último, en las pruebas realizadas en [10], se incluye un algoritmo voraz multiarranque MSGA denominado MSG.

4.3. Especificación del problema

Entre las soluciones encontradas en el estado del arte, y que servirán de base para el desarrollo que se presentará en esta memoria, se encuentran las siguientes trabajos (ver sección 4.2.4 para más información):

- Algoritmo de colonia de hormigas (ACO) propuesto en [33] y [34], así como un algoritmo voraz multiarranque (MSGA).
- Los métodos LS-SGA y LS-R según el camino óptimo restringido (COP) de [80].
- Algoritmo genético (GA) [120].
- Los algoritmos de paso de mensajes MP y MP_*reinf* [10]; éste último con aprendizaje por refuerzo. Además se incluye un algoritmo voraz multiarranque MSG.

Por tanto, habrá que adaptar algunas particularidades propias del problema a tratar para que, una vez resuelto, se pueda realizar una comparación objetiva con las soluciones previas:

- Primeramente, hay que destacar que los grafos del problema planteado tienen arcos con capacidad 1. Es decir, dichos arcos sólo tienen posibilidad de formar parte de un único camino a la vez. Los nodos no tienen restricciones al respecto.
- Por otra parte, los arcos tienen distancia 1. Es decir, que si se utilizase el criterio de distancia mínima entre nodos, la distancia se medirá como el número de arcos (o nodos) del camino.
- Los grafos utilizados son no dirigidos.

Atributos a optimizar		
Atributo	Valor	Porcentaje
Rendimiento	84	44 %
Coste	74	39 %
Fiabilidad	71	37 %
Disponibilidad	25	13 %
General	22	12 %
Peso	5	3 %
Seguridad (*safety*)	4	2 %
Reputación	4	2 %
Adaptabilidad	3	2 %
Protección (*security*)	1	< 1 %

Tabla 4.1: Relación de los aspectos generales a optimizar en los problemas según el estudio descrito en [8].

Clasificación del problema de optimización

En 2013, en el trabajo [8] se realizó un estudio general del uso de las técnicas de optimización en la resolución de problemas *software*. Se analizaron 188 artículos, con el objetivo de dar a conocer una visión general de diferentes aspectos relacionados con los problemas que han resuelto, como pueden ser el dominio del problema, los parámetros a optimizar, o las técnicas de resolución aplicadas. Este análisis puede ser muy interesante para clasificar el problema EDP dentro del conjunto de problemas resueltos por los métodos de optimización.

Hay que destacar que las categorías expuestas en las tablas no son excluyentes. Es decir, que el problema o solución expuesto en un artículo analizado podría pertenecer a diferentes clases.

En la tabla 4.1[6] se clasifican los problemas estudiados según los diversos parámetros generales a optimizar. Las categorias más frecuentes son el rendimiento, el coste y la fiabilidad. El problema EDP, teniendo en cuenta que maximiza el número de caminos, podría incluirse dentro de las clases de rendimiento y fiabilidad. En cualquier caso, categorías que incluyen numerosos problemas de optimización.

En la tabla 4.2, incluida también en el trabajo [8], aparecen las subcategorías relacionadas con los aspectos descritos en la tabla 4.1, detallando el objetivo de los diferentes ejercicios de

[6] *safety* se refiere a la seguridad de los procesos, mientras que *security* a la protección de los sistemas

Subcategorías de optimización		
Atributo	Valor	Porcentaje
Asignación	59	31 %
Replicación de *hadware*	40	21 %
Selección de *hadware*	38	20 %
Replicación de *software*	35	18 %
Planificación	33	17 %
Selección de componentes	30	16 %
Selección del servicio	28	15 %
Selección del *software*	28	15 %
Otros problemas específicos	18	9 %
Composición del servicio	12	6 %
Parámetros de *software*	10	5 %
clustering	5	3 %
General *software*	5	3 %
Parámetros de *hardware*	4	2 %
Patrones arquitectónicos	3	2 %
No existen	3	2 %
Particiones	2	1 %
Plan de mantenimiento	2	1 %

Tabla 4.2: Subcategorías de optimización.

Dominio del problema		
Atributo	Valor	Porcentaje
Sistemas embebidos	100	53 %
Genéricos	49	26 %
Sistemas de gestión empresarial	41	22 %

Tabla 4.3: Dominios de los problemas.

optimización analizados. EDP podría corresponderse con varias clases, como podrían ser la de asignación (de caminos que comunican pares terminales) o la de selección de enlaces físicos (tanto *hadware* como de componentes).

Dimensión del problema		
Atributo	Valor	Porcentaje
Mono-objetivo	75	41 %
Multi-objetivo	109	59 %

Tabla 4.4: Dimensión del problema.

En la tabla 4.3 aparecen las categorías de dominios asociados a los problemas analizados en el estudio [8]. Éstos podrían ser sistemas empotrados o embebidos, sistemas de gestión empresarial o genéricos. En el caso del problema EDP, no estaría clara su categoría, por lo que se incluiría en la clase genérica.

Mientras que el problema EDP a resolver tiene el objetivo fundamental de maximizar el número de caminos disjuntos (aunque también hay que tener en cuenta el tiempo de cómputo), se aprecia en el estudio desarrollado en [8] que la proporción de ejercicios multiobjetivo

Resultados de la propuesta		
Atributo	Valor	Porcentaje
Experimentos	57	30 %
Ejemplo sencillo	51	27 %
Caso de estudio académico	31	16 %
Caso de estudio industrial	30	16 %
No presentado	19	10 %
Benchmark	8	4 %
Comparación con la "literatura" (estado del arte)	2	1 %
Demostración matemática sencillo	1	< 1 %

Tabla 4.5: Validación de los resultados de las soluciones desarrolladas.

analizados es mayor (ver tabla 4.4). A pesar de lo anterior, no existe mucha diferencia entre ambos valores (41 % de problemas mono-objetivo frente a 59 % de multi-objetivos).

		Estrategia de optimización				
Problema		Métodos exactos		Métodos aproximados		
Atributo	Valor	Estándar	Específ. del problema	Metaheur.	Heur. específ. del problema	Otros
Rendimiento	84	14 (17 %)	3 (4 %)	45 (54 %)	23 (27 %)	4 (5 %)
Coste	84	13 (18 %)	2 (3 %)	45 (61 %)	13 (18 %)	4 (5 %)
Fiabilidad	71	7 (10 %)	2 (3 %)	49 (69 %)	11 (15 %)	4 (6 %)
General	25	1 (4 %)	2 (8 %)	10 (40 %)	4 (16 %)	6 (24 %)
Disponibilidad	22	7 (32 %)	-	12 (55 %)	7 (32 %)	2 (9 %)
Energía	18	4 (22 %)	2 (11 %)	6 (33 %)	7 (39 %)	2 (12 %)
Peso	5	-	-	5 (100 %)	-	-
Seguridad (*safety*)	4	-	-	3 (75 %)	-	1 (25 %)
Reputación	4	2 (50 %)	-	1 (25 %)	2 (50 %)	-
Adaptabilidad	3	-	-	3 (100 %)	-	-
Protección (*security*)	1	-	-	1 (100 %)	-	-

Tabla 4.6: Asignación de estrategias de optimización a los diferentes problemas.

En la tabla 4.5 aparece la manera en las que se han desarrollado las pruebas sobre las diferentes propuestas analizadas en [8]. Como se mencionará en el posterior capítulo de resultados (6), en el caso presentado en esta memoria, se hará uso de las pruebas de rendimiento sobre los *benchmarks* empleados en [34], [80], [120] y [10]; aunque también se utilizarán otros ejemplos sencillos cuya finalidad es la consecución de pruebas unitarias y de funcionalidades. Sobre dicha tabla (4.5), hay que destacar la poca proporción de artículos que presentan soluciones con *benchmarks* o comparaciones con el estado del arte. Es decir, que en muchos casos, no se podría conocer de forma objetiva, si la solución propuesta mejora las estrategias previas de optimización (si las hubiera).

Una tabla muy importante, que muestra la aplicación de las diferentes estrategias de optimización a diferentes problemas analizados en el trabajo [8] es la tabla 4.6. Como se puede apreciar, por un lado, se presentan diferentes categorías del problema, que indican la

característica que se desea optimizar. Por otra parte, dentro de las estrategias de optimización, se encuentran los métodos exactos, aproximados y otros. Dentro de los exactos, aparecen los estándar y específicos para el problema. Los aproximados de clasifican en metaheurísticas y heurísticas específicas del problema. Por último, en otros, se incluyen los enfoques generales o no presentados. Claramente aparecen las metaheurísticas como la solución más utilizada en la resolución de los diferentes problemas, seguidas de las heurísticas específicas y los métodos exactos estándar. Esta información muestra pistas de las tendencias que se han seguido en la resolución de diversos problemas de optimización.

4.3.1. Hipótesis

Una vez identificado el problema que se pretende resolver, el desarrollo de la investigación prosigue con la formulación de una hipótesis inicial. Dicha conjetura es una propuesta tentativa que pretende proponer una solución al problema abordado. La hipótesis representaría un elemento fundamental en el proceso de resolución del problema, ya que serviría de guía al mismo.

La conjetura que se plantea para el desarrollo de este documento se puede resumir en lo siguiente: el problema de los caminos disjuntos es un problema clásico NP, con interés práctico en distintas disciplinas científicas e ingenieriles. Por ello, es interesante el desarrollo de algoritmos que sean capaces de resolverlo de forma eficiente, obteniendo soluciones óptimas o, en su defecto, de alta calidad, en el menor tiempo posible.

Para la resolución de este tipo de problemas son relevantes tanto los algoritmos de naturaleza heurística, como exacta. Los primeros, son capaces de obtener soluciones de alta calidad en un tiempo razonable. Por otro lado, los métodos exactos, son capaces de resolver el problema de manera óptima, cuando el tamaño de la instancia es reducido.

Tal y como se comenta en la sección 3.4, dedicado a las matheurísticas, los métodos exactos presentan ciertas ventajas como pueden ser: la obtención de resultados óptimos, si el algoritmo tiene éxito en su ejecución; o de cotas superiores y/o inferiores de la solución, si el proceso se detiene antes de completar su espacio de búsqueda. Debido al rastreo exahustivo que suelen llevar a cabo en el espacio de soluciones, aparecen ciertos inconvenientes cuando las instancias del problema y el espacio de prospección empiezan a aumentar. En estos casos el tiempo necesario para alcanzar una solución competitiva puede ser inabordable.

Por otra parte, los procedimientos metaheurísticos presentan ciertas ventajas como pueden ser la facilidad de su implementación, asi como la flexibilidad de adaptación a diferentes variantes de un problema. Pero también sufren inconvenientes: no pueden asegurar ni la optimalidad ni la convergencia en la solución (el resultado calculado podría situarse "lejos" de la mejor solución posible).

Por tanto, existen tres estrategias fundamentales para solventar un problema NP:

1. Si la entrada es pequeña, un algoritmo con tiempo de ejecución exponencial podría ser perfectamente aceptable.

2. Se podrían aislar algunos casos especiales que resolviesen el problema en tiempo polinomial.

3. Se permitirían utilizar aproximaciones para encontrar soluciones lo suficientemente cercanas al óptimo en tiempo polinomial.

En la práctica, obtener soluciones cercanas al óptimo es bastante aceptable. En muchos casos, estos procesos se apoyan en heurísticas y metaheurísticas.

Debido a que no va a existir ninguna limitación en los parámetros de entrada del problema. Es decir, que los grafos de entrada y conjuntos de pares terminales serán de cualquier dimensión; así como grafos de cualquier topología, se deberían descartar a priori las estrategias 1 y 2.

Pero, a partir de los planteamientos anteriores, se podría pensar en algún proceso candidato que se pudiese comportar de forma eficiente para cualquier tipo de entrada. Es decir, que para datos de entrada de pequeñas dimensiones o topología de grafos sencillos, se pudiesen obtener soluciones exactas en un tiempo muy limitado. Y en cambio, con grafos de topología más compleja y de mayor magnitud, así como conjunto de pares terminales de gran tamaño, se pudiesen obtener resultados relativamente cercanos al óptimo.

Una colaboración entre algoritmos exactos y metaheurísticos podría proporcionar métodos híbridos que englobasen las ventajas y fortalezas de cada uno de ellos. A estos procedimientos mixtos se les llaman matheurísticas. Por tanto, éstos últimos deberían superar a ambos procedimientos (exacto y metaheurístico) cuando se empleen por separado.

Asímismo, se podría finalizar estableciendo como hipótesis que el enfoque matheurístico obtendría mejores resultados, en un menor tiempo de cómputo, que las propuestas presentadas en [34], [80], [120] y [10].

4.3.2. Objetivos

Los objetivos de esta investigación se derivan directamente de la hipótesis enunciada anteriormente y consisten en los siguientes puntos:

- Desarrollar algoritmos de resolución para el problema de los caminos disjuntos, desde el punto de vista matheurístico, con una combinación de un modelo exacto y un procedimiento heurístico / metaheurístico.
- Alcanzar una mejora de los resultados obtenidos por los artículos de referencia, mediante el procedimiento desarrollado.
- Por último, es importante tener en cuenta que el procedimiento a desarrollar, además de alcanzar las mejores soluciones, deberá ser original; es decir, no implementado y publicado anteriormente.

Para alcanzar los objetivos descritos en las líneas anteriores, es necesario cubrir los siguientes propósitos parciales:

- Conocer el estado del arte del problema. Se debe realizar un estudio bibliográfico de la cuestión, recopilando las diferentes técnicas utilizadas para la resolución del mismo, tanto con algoritmos heurísticos como con métodos exactos. Además, se tomarán los conjuntos de instancias (*benchmarks*), sobre los cuales han sido evaluados dichos algoritmos.
- Desarrollar un modelo exacto para la resolución del ejercicio.

- Diseñar e implementar un algoritmo metaheurístico para la resolución de la cuestión.
- El matheurístico debe alcanzar mejores resultados que los obtenidos por el exacto y el metaheurístico cuando ambos se utilizan de forma aislada.
- Comparar experimentalmente los procedimientos desarrollados con el estado del arte. Más exactamente, se contrastarán los resultados de función objetivo y tiempo de ejecución de los algoritmos desarrollados, con los presentados en [34], [80], [120] y [10].
- Redactar una memoria que incorpore los resultados de investigación.
- Someter los resultados parciales y totales del trabajo desarrollado a procesos de revisión por parte de instituciones independientes. Los estudios obtenidos serán enviados a congresos y revistas de reconocido prestigio nacional e internacional.

4.3.3. Método de investigación

El plan utilizado para el desarrollo de esta tesis propone una serie de etapas, que por su generalidad, son aplicables con ligeras modificaciones a cualquier tipo de investigación [232]:

1. Análisis del problema.
   a) Conocimiento en profundidad del problema y sus variantes.
   b) Revisión de métodos de resolución existentes en el estado del arte.
   c) Formulación de hipótesis y definición de objetivos
2. Diseño del algoritmo.
   a) Selección de la estrategia de resolución.
   b) Definición de las medidas de rendimiento.
   c) Selección de las estructuras de datos.
   d) Especificación del algoritmo.
3. Implementación.
   a) Implementación del algoritmo.
   b) Ajuste de parámetros (*tuning*).
   c) Análisis del rendimiento del algoritmo.
   d) Informe de resultados y comparación con otros métodos propuestos.

Una vez identificado el ejercicio de optimización que se pretende resolver, el proceso comienza con una revisión del estado del arte del mismo (capítulos 2 y 3 y sección 4.2 del documento). Como resultado de dicha exploración, se identifican los algoritmos previamente desarrollados para el problema en cuestión, así como los métodos y técnicas que se pueden emplear para resolverlo.

A partir de las tareas anteriores, se plantea una hipótesis y se marcan los objetivos (sección 4.3 del documento) que orientarán la resolución del problema, y que desembocarán en el desarrollo del mismo (capítulo 5).

La validación de los procedimientos implementados se realiza en distintas etapas. Para ello, se emplea un conjunto de instancias de referencia que permitan comparar los algoritmos con los presentados en el estado del arte. Asímismo se realizará un ajuste de diferentes parámetros para mejorar el comportamiento del mismo, y un análisis comparativo con los métodos propuestos anteriormente. Este proceso finaliza con un informe de resultados. Todas estas tareas se describen en el capítulo 6 y en el anexo A del documento.

Por último, es importante destacar que todo el proceso descrito es iterativo. De esta manera, ante cualquier necesidad, desde cualquier etapa se puede saltar a otra anterior, y proseguir con el proceso habitual.

Capítulo 5

Método de resolución

5.1. Introducción

Como se ha comentado en el capítulo anterior, y más exactamente en las hipótesis y objetivos de la tarea a emprender, se debe plantear y desarrollar una solución inédita que mejore el rendimiento alcanzado en trabajos previos presentados en [34], [80], [120] y [10].

En las siguientes páginas se detallará el proceso propuesto que resuelve el problema EDP. Se basa en una matheurística simple de dos etapas: por una parte se tiene un modelo exacto ILP, que calcula un resultado al problema en un tiempo limitado. Si se alcanza la solución óptima, el proceso finaliza. En caso contrario, aparece una metaheurística poblacional, que toma el resultado calculado por el componente anterior e intentará mejorarlo, también en un tiempo de ejecución restringido. Este último componente es un algoritmo multiagente, en la que cada individuo de la población se podría comportar de manera autónoma. Además, se incluyen mecanismos de colaboración entre cada entidad de la población basados (aunque sea de manera muy sutil) en el método PSO.

Esta estructura *software* debe incluir las ventajas de ambos componentes. Por un lado, el modelo ILP encuentra un buen resultado, incluso el óptimo, si tiene suficiente tiempo para hacerlo. Por otro lado, el metaheurístico puede calcular buenos resultados en tiempo acotado.

No se ha encontrado en el estado del arte ningún proceso matheurístico que resuelva el problema EDP de esta manera. Se ha localizado algún artículo que utiliza un algoritmo PSO para resolver el problema de la enrutación dinámica y asignación de longitud de onda (Dynamic Routing and Wavelenght Assigment -RWA-) [116]. Éste es un problema parecido al EDP, pero tienen objetivos diferentes: RWA trata de minimizar el número de longitudes de onda que comparten los arcos en el grafo (ver explicación del ejercicio en [9]). En cualquier caso, se conectan todos los nodos terminales del problema; a diferencia del problema EDP, donde se podrían calcular resultados en los que podrían existir pares terminales sin conectar. Por otra parte, el proceso híbrido planteado incluye algunos conceptos de cooperación del método PSO en el algoritmo poblacional; pero en ningún caso se puede considerar el metaheurístico como un algoritmo PSO. Por tanto, se puede cumplir con uno de los requerimientos de la solución propuesta: que sea singular, en el sentido de que no se haya empleado previamente para la

resolución del problema EDP un algoritmo similar al especificado en las siguientes secciones del capítulo.

5.2. Una visión global

La estructura de optimización propuesta tiene como objetivo el cálculo de resultados en un tiempo limitado. Está compuesta por un modelo de programación matemática (ILP) y un algoritmo metaheurístico (MH), ambos interconectados, tal y como aparece en la figura 5.1.

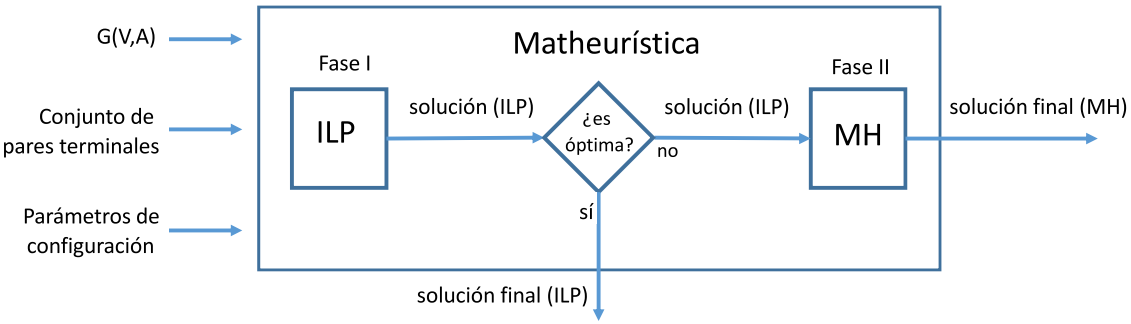

Figura 5.1: Estructura general del algoritmo híbrido a desarrollar.

Tomando como punto de partida que una matheurística es una hibridación de un método exacto y otro metaheurístico, parece que la estructura de dos etapas indicada anteriormente podría ser perfectamente catalogada como una matheurística colaborativa secuencial. En este caso, el modelo ILP y el procedimiento MH se ejecutan de forma continua. Primeramente el exacto, y cuando finaliza éste, el algoritmo poblacional, tal y como se presenta en la clasificación descrita en [196]. Según la clasificación de [127], este proceso de dos pasos que se presenta en el documento se correspondería con la categoría HRH. Se recuerda que en esta clase de organización algorítmica, los diferentes procedimientos, que son autónomos y autocontenidos, se ejecutan igualmente de forma secuencial.

Criterios de diseño.

Además de pertenecer a la categoría HRH, la solución propuesta tiene las siguientes características de diseño (existe más información en la sección 3.4.4 del capítulo 3.4):

- Resolución exacta y aproximada del ejercicio. Si únicamente es necesario ejecutar la primera fase del proceso, el resultado será óptimo; por lo que el método llevará a cabo una resolución exacta del problema EDP. Si se incorpora a la anterior la segunda etapa del procedimiento híbrido, se obtendrá un resultado final aproximado, debido a que no se puede asegurar que la salida del proceso global alcance un valor óptimo del problema.
- Se emplea una cooperación de búsqueda global. Los dos componentes del matheurístico actúan en todo el espacio de búsqueda.
- Naturaleza de cooperación general. Los métodos ILP y MH resuelven el mismo problema de optimización,

Criterio de implementación.

Claramente la implementación propuesta es secuencial. Debe finalizar el modelo exacto para poder decidir, dependiendo de si el resultado alcanzado es óptimo o no, sobre la la necesidad de arrancar la fase del MH.

Comportamiento del sistema.

Por tanto, tal y como se presenta en la figura 5.1, la matheurística debe tomar como parámetros de entrada el grafo del ejercicio, el conjunto de pares terminales a conectar, y un conjunto de parámetros entre los que se pueden encontrar algunos de tipo general, como puede ser el tiempo de ejecución total del proceso, y el porcentaje de dicho tiempo repartido entre los dos componentes de la estructura híbrida; y otros atributos más específicos, como el número de agentes que contiene el MH.

Primeramente, el método exacto calcula un resultado en el tiempo asignado para ello, comprobando si dicha solución es óptima. En caso afirmativo, se devuelve directamente dicho resultado y finaliza el proceso. En otro caso, la salida del ILP es tomada por MH como entrada al algoritmo, con el objetivo de alcanzar una solución final mejorada. Para llevar a cabo esta tarea, el metaheurístico también tiene asignado un tiempo máximo.

Para cumplir con las expectativas de este trabajo, recogidas en las hipótesis del problema, los algoritmos exacto y heurístico incluidos en el matheurístico deben superan a ambos procedimientos cuando se tratan de forma independiente por separado. Desde el punto de vista del MH, el resultado del ILP proporciona un buen punto de partida del algoritmo poblacional; y desde el punto de vista del exacto, el procedimiento MH debera mejorar, en tiempos ajustados, las soluciones del ILP que se encuentren por debajo del óptimo. Por lo tanto, ambas estrategias pueden considerarse complementarias, y su hibridación puede proporcionar métodos efectivos en la resolución del problema.

5.3. El modelo de programación entera

5.3.1. Relajación del problema

El problema EDP se podría modelar a partir de una formulación del problema de flujo *multicommodity* [143] basado en rutas. Debido a que se permite, en este caso, flujos fraccionales, se correspondería con un problema P; pudiendo ser, por tanto, resuelto mediante una formulación LP. Las siguientes expresiones matemáticas representan el modelo que resuelve el problema *Multicommodity Flow*. Sin embargo, requiere un número exponencial de variables dependiendo del tamaño del grafo del problema. De esta manera, suponiendo que:

$\mathcal{G}(\mathcal{V}, \mathcal{E})$ es el grafo del problema

Se desea conectar un conjunto de K pares terminales $\{(s_1, t_1), ..., (s_i, t_i), ...(s_k, t_k)\}$

$\mathcal{P}_i$ es el conjunto de rutas que comunican los pares terminales s_i hasta t_i, y

$\mathcal{P}$ es el conjunto de caminos que comunican todos los pares terminales $\mathcal{P} = \bigcup_{i=1}^{k} \mathcal{P}_i$

Se tiene el siguiente modelo:

$$\text{máx} \quad \sum_{P \in \mathcal{P}} f_P \tag{5.1}$$

$$\text{sujeto a} \quad \sum_{P \in \mathcal{P}_i} f_P \leq d_i \qquad i = 1, ..., k \tag{5.2}$$

$$\sum_{P \in \mathcal{P}, e \in P} f_P \leq u_e \qquad e \in E \tag{5.3}$$

$$f_p \geq 0 \qquad P \in \mathcal{P} \tag{5.4}$$

En este caso, además de capacidad de arcos, se debe satisfacer la demanda d_i de cada par terminal i (ecuación 5.2). El objetivo es, por tanto, maximizar la suma de las demandas que pueden ser enrutadas (ecuación 5.1), cumpliendo con las restricciones de capacidad u_e para cada enlace $e \in E$ (ecuación 5.3). Como se ha comentado en líneas anteriores, en este modelo se permiten flujos fraccionales, compartidos en diversas rutas para cada par terminal (ecuación 5.2), por lo que la resolución del problema sería de complejidad P.

Si la demanda de cada par terminal d_i es igual a 1 y la capacidad máxima de cada arco del grafo también contiene un valor de 1, el problema se podría considerar como un EDP generalizado (*generalized MEDP)*, que se puede considerar como una relajación del EDP.

En la siguiente sección se mostrará un modelo ILP que resuelve el problema EDP, y que requiere un conjunto polinómico de variables en relación con el grafo del ejercicio.

5.3.2. Formulación del modelo ILP

A diferencia del modelo anterior, que necesita un número de variables exponencial respecto al tamaño del grafo, en este documento se propone una formulación binaria de flujo de red *multi-commodity* del problema EDP que necesita únicamente un conjunto de variables de magnitud polinomial en relación a la dimensión del grafo.

Pensando en términos de flujo, el problema EDP se centra en enviar una sola unidad de flujo desde el nodo inicial i_k al nodo final j_k, para todos los pares terminales $[i_k, j_k] \in \mathcal{T}$. De esta forma, el flujo del par terminal k crea una ruta desde el nodo i_k al nodo j_k asociado al par terminal k, para todos los $k \in \mathcal{K} = \{1, \ldots, K\}$.

Se ha empleado la notación que se muestra en la tabla 5.1. Básicamente, los nodos del grafo se clasifican en tres grupos: nodos de suministro -iniciales- ($\mathcal{V}^-$), de demanda -finales- ($\mathcal{V}^+$) e intermedios ($\mathcal{V}^0$). Los nodos del primer grupo se etiquetan según el índice del par terminal correspondiente k; y los del segundo conjunto se encuentran marcados por el indicativo $K + k$. Con esta anotación, cada par terminal $[i_k, j_k]$ se etiqueta como $[k, K + k]$ para todos $k \leq K$. Es necesario indicar que cualquier nodo $v \in \mathcal{V}^-$ es un nodo de suministro para el par terminal $k = v$, pero asímismo puede actuar con un rol de nodo intermedios para otros pares terminales (ver ecuación 5.8). De forma similar, cualquier nodo $v \in \mathcal{V}^+$ es un nodo de demanda para el par terminal $k = v - K$, pero también es un nodo intermedio para otros pares terminales (ver ecuación 5.12).

El modelo ILP se definiría con la siguiente formulación:

Parámetros	Valores	Descripción	
$\mathcal{V}$	$\{1,\ldots,K,\ldots,V\}$	Conjunto total de nodos	
$\mathcal{V}^-$	$\{1,\ldots,K\}$	Nodos iniciales (origen, fuente o de suministro)	
$\mathcal{V}^+$	$\{K+1,\ldots,K+K\}$	Nodos finales (destino o de demanda)	
$\mathcal{V}^0$	$\mathcal{V}\setminus(\mathcal{V}^-\cup\mathcal{V}^+)$	Nodos intermedios	
i, j, v	-	Índices de los nodos	$i,j,v\in\mathcal{V}$
$\mathcal{E}$	-	Conjunto de enlaces	
(i,j)	-	Índice de los nodos conectados por un enlace	$(i,j)\in\mathcal{E}$
$\mathcal{G}$	$(\mathcal{V},\mathcal{E})$	Grafo no dirigido	
$\mathcal{K}$	$\{1,\ldots,K\}$	Conjunto de pares terminales	
k	-	Índice de los pares terminales	$k\in\mathcal{K}$
x_{ijk}	$\{0,1\}$	Flujo correspodiente al par terminal k que circula entre los nodos i y j	$i,j\in\mathcal{V},\ k\in\mathcal{K}$

Tabla 5.1: Notación de la formulación ILP.

$$\max_{x} \quad z_{EDP} = \sum_{v\in\mathcal{V}^-}\sum_{(v,j)\in\mathcal{E}} x_{vjv} \tag{5.5}$$

$$\text{sujeto a} \quad \sum_{(v,j)\in\mathcal{E}} x_{vjv} \le 1 \qquad v\in\mathcal{V}^- \tag{5.6}$$

$$\sum_{(i,v)\in\mathcal{E}} x_{ivv} = 0 \qquad v\in\mathcal{V}^- \tag{5.7}$$

$$\sum_{(i,v)\in\mathcal{E}} x_{ivk} - \sum_{(v,j)\in\mathcal{E}} x_{vjk} = 0 \qquad v\in\mathcal{V}^-,\quad k\in\mathcal{K}\setminus\{v\} \tag{5.8}$$

$$\sum_{(i,v)\in\mathcal{E}} x_{ivk} - \sum_{(v,j)\in\mathcal{E}} x_{vjk} = 0 \qquad v\in\mathcal{V}^0,\quad k\in\mathcal{K} \tag{5.9}$$

$$\sum_{(i,v)\in\mathcal{E}} x_{ivk} \le 1 \qquad v\in\mathcal{V}^+,\quad k=v-K \tag{5.10}$$

$$\sum_{(v,j)\in\mathcal{E}} x_{vjk} = 0 \qquad v\in\mathcal{V}^+,\quad k=v-K \tag{5.11}$$

$$\sum_{(i,v)\in\mathcal{E}} x_{ivk} - \sum_{(v,j)\in\mathcal{E}} x_{vjk} = 0 \qquad v\in\mathcal{V}^+,\quad k\in\mathcal{K}\setminus\{v-K\} \tag{5.12}$$

$$\sum_{k\in\mathcal{K}} x_{ijk} + \sum_{k\in\mathcal{K}} x_{jik} \le 1 \qquad (i,j)\in\mathcal{E},\quad i<j \tag{5.13}$$

$$x_{ijk}\in\{0,1\} \qquad (i,j)\in\mathcal{E},\quad k\in\mathcal{K}, \tag{5.14}$$

y se podrían incluir los siguientes comentarios:

Eq. (5.5): Se maximiza el flujo total enviado desde los nodos fuente. Este valor es equivalente a alcanzar el mayor número de caminos disjuntos.

Como se muestra en la figura 5.2, el cálculo de los caminos disjuntos se computan

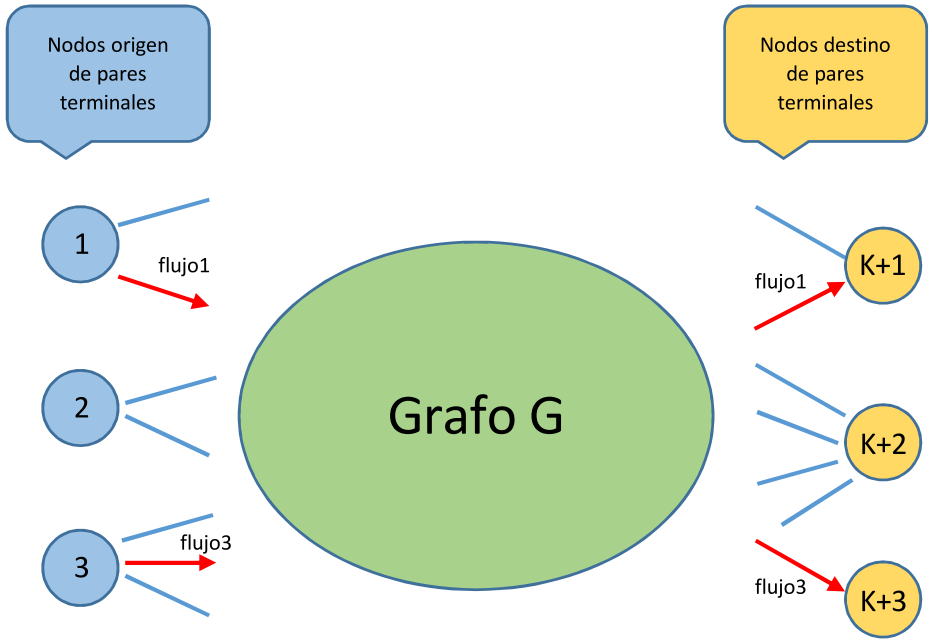

Figura 5.2: Cálculo de la función objetivo del modelo.

obteniendo los flujos que parten de los nodos origen. También se podría haber utilizado para obtener la función objetivo del problema, las rutas que finalizan en los nodos destino.

Eq. (5.6): Cada nodo fuente o inicial v puede enviar a lo sumo una unidad de flujo para el par terminal v.

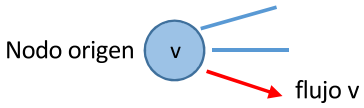

Figura 5.3: Restricción en los nodos origen.

Esto indica simplemente que desde cada nodo origen v sólo puede partir, a lo sumo, un arco que pertenezca a la solución del par terminal v.

Eq. (5.7): Cada nodo inicial v no puede recibir flujo del par terminal v. Esta sentencia evita la generación de ciclos en caminos que comiencen y terminen en nodos de demanda.

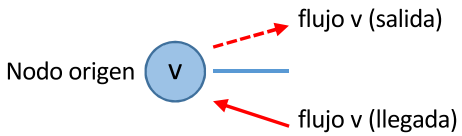

Figura 5.4: Ciclos en los nodos origen.

Como se aprecia en la figura 5.4, el camino del par terminal v no puede finalizar en el nodo v (teniendo conocimiento que, además, la ruta, si existe, ha partido de dicho nodo).

Eq. (5.8): En cada nodo fuente v de cada par terminal k, el flujo de entrada es el mismo que el de salida, para todos los $k \in \mathcal{K} \backslash \{v\}$.

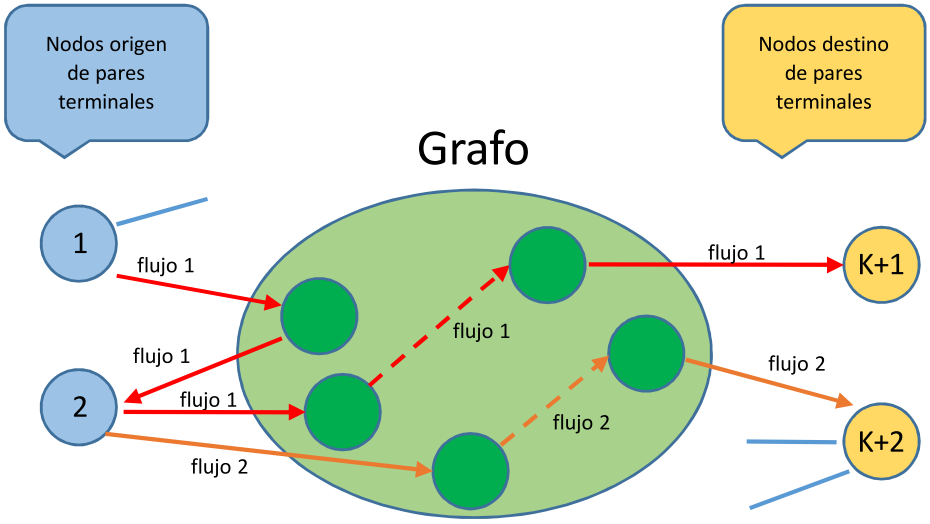

Figura 5.5: Nodo origen desempeñando labor de intermediario.

En la figura 5.5 se puede apreciar al nodo 2 ejerciendo de nodo intermedio para el camino 1. Sin la restricción incluida en la ecuación (5.8), la ruta 1 podría finalizar en el nodo 2. Por otra parte, lo anterior no impide que el propio nodo 2 pueda lanzar el flujo 2 con destino final $K + 2$

Eq. (5.9): En todos los nodos intermedios, el flujo de entrada es el mismo que el de salida.

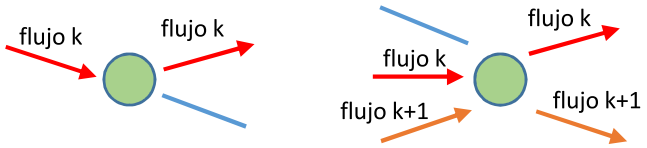

Figura 5.6: Restricción en nodos intermedios. El flujo tiene que ser cero.

Por cada nodo intermedio puede pasar cualquier número de caminos que pertenezcan a la solución (ver figura 5.6). Pero estos nodos no permiten que comience o finalice una ruta en él (ver figura 5.7).

Figura 5.7: Restricción en nodos intermedios. Casos no contemplados.

Eq. (5.10): Cada nodo destino v puede recibir a lo sumo una unidad de flujo del par terminal $(v - K)$. Es necesario destacar que el nodo final $v = K + k$ recibe el flujo del par terminal k, tal que $k = v - K$.

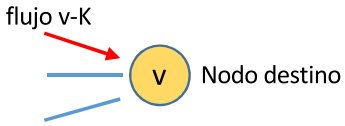

Figura 5.8: Restricción en los nodos destino.

Tal y como se representa en la figura 5.8, a cada nodo destino v sólo puede llegar, a lo sumo, un arco que pertenezca al camino solución del par terminal $v - K$.

Eq. (5.11): Cada nodo destino v no puede enviar flujo para el par terminal $(v - K)$.

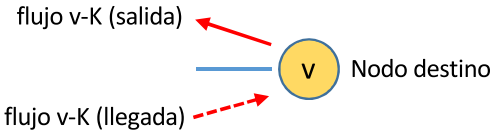

Figura 5.9: Ciclos en los nodos destino.

Como se aprecia en la figura 5.9, el camino del par terminal $v - K$ no puede comenzar en el nodo v. Además la ruta, si existe, ha finalizado de dicho nodo.

Eq. (5.12): En cada nodo destino v y para cada par terminal k, el flujo de entrada es igual al de salida, para todo $k \in \mathcal{K} \backslash \{v - K\}$.

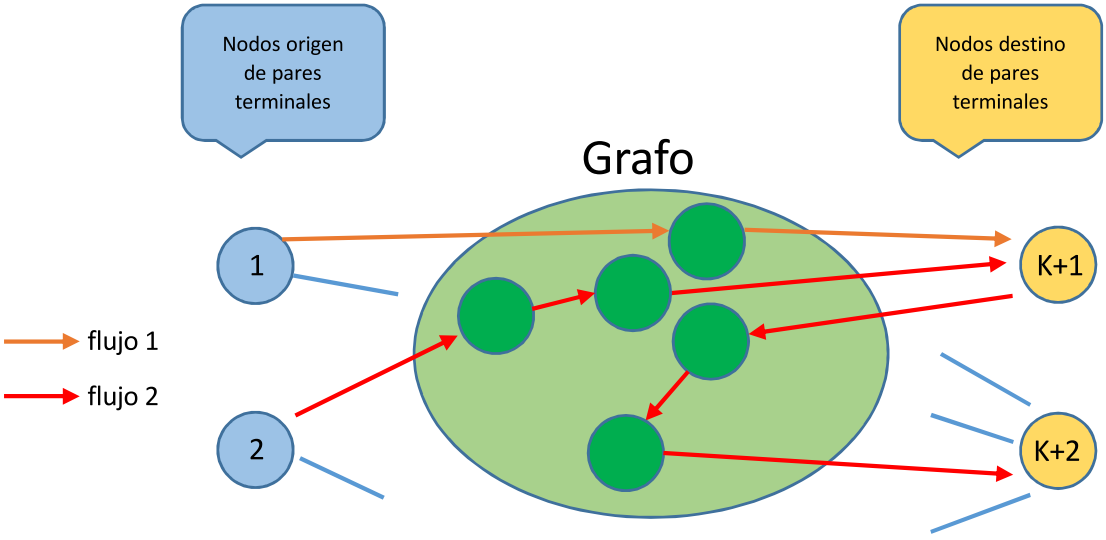

Figura 5.10: Nodo destino desempeñando labor de intermediario.

En la figura 5.10 aparece el nodo K+1 ejerciendo de nodo intermedio para el camino 2; sin entorpecer que el propio nodo 1 ejerza de destino del camino 1.

Eq. (5.13): La capacidad de cada arco del grafo es 1. Hay que indicar que los flujos que conectan los pares terminales pueden hace uso de cada enlace $(i, j) \in \mathcal{E}$, en cualquiera de las dos direcciones, debido a que son no dirigidos. Pero no en ambos a la vez.

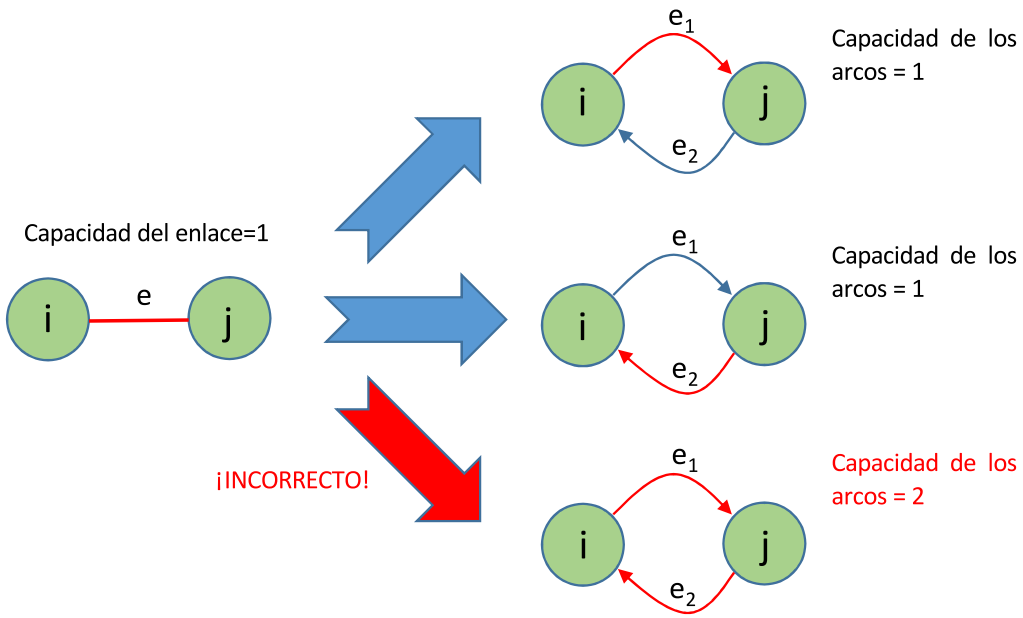

Figura 5.11: Capacidad de los enlaces del grafo.

La figura 5.11 señala que los arcos del grafo se pueden utilizar para únicamente un camino a lo sumo, independientemente de que las rutas tomen los arcos en las mismos o diferentes sentidos.

Validez del modelo

La validez de la formulación (5.5)-(5.14) para el problema EDP se puede tratar de la siguiente manera:

Primeramente, dado un nodo fuente $\bar{v} \in \mathcal{V}$ y un enlace $(\bar{v}, j) \in \mathcal{E}$, si se tiene $x_{\bar{v}j\bar{v}} = 1$, existe una ruta que conecta el par de nodos terminales $[\bar{v}, K + \bar{v}]$. De forma más detallada, la igualdad $x_{\bar{v}j\bar{v}} = 1$ significa que el nodo origen $\bar{v}$ envía una unidad en el flujo $\bar{v}$ al nodo j. Las ecuaciones (5.8), (5.9) y (5.12) exigen que todos los nodos del grafo se comporten como nodos intermedios para el flujo $\bar{v}$, excepto para su nodo fuente $\bar{v}$ y receptor correspondiente $K + \bar{v}$. Asímismo, la ecuación (5.7) no permite que en el nodo fuente $\bar{v}$ pueda ser destino de un flujo del par terminal $\bar{v}$. Esto impide, por un lado, que se creen ciclos de flujo situados en nodos origen; y por otra parte, que el flujo pueda finalizar en dichos nodos. De forma similar la ecuación (5.11) no permite que el nodo destino $\bar{v}$, pueda ser origen del flujo $v - K$. Esto impide la generación de ciclos en nodos finales. El número de enlaces del grafo es finito y cada uno de ellos admite, a lo sumo, una unidad de flujo. Por lo tanto, esta transmisión de flujo iniciada en el nodo origen $\bar{v}$ terminará necesariamente en el nodo receptor $K + \bar{v}$. Como consecuencia, se tendrá $x_{ivk} = 1$ por algunos $i \in \mathcal{V}$, para $v = K + \bar{v}$ y para $k = \bar{v}$. El flujo $\bar{v}$ que se acaba de describir define una ruta que conecta el par de nodos terminales $[\bar{v}, K + \bar{v}]$. Es decir, que si un flujo $k = \bar{v}$ sale desde su nodo origen $k = \bar{v}$, necesariamente debe finalizar en el nodo destino $K + \bar{v}$. En otro caso, desde en nodo fuente $k = \bar{v}$ no circulará ningún flujo.

Segundo, dados dos nodos fuente $\bar{u}$ y $\bar{v}$, si se tiene $x_{\bar{u}i\bar{u}} = x_{\bar{v}j\bar{v}} = 1$ entonces las rutas

que conectan los pares terminales correspondientes son disjuntos. Esta sentencia se afirma directamente mediante la ecuación (5.13), relativa a las capacidades de los arcos.

Definiendo EDP^* como el número máximo de caminos disjuntos asociados al par $(\mathcal{G}, \mathcal{T})$; y considerando z^*_{EDP} como el valor óptimo del modelo ILP correspondiente (5.5)-(5.14), se puede afirmar que $z^*_{EDP} \leq EDP^*$.

Tercero, hay que comprobar si $EDP^* \leq z^*_{EDP}$: dado un par de nodos terminales $[\bar{v}, K + \bar{v}]$, conectados por un ruta con arcos disjuntos P, es posible enviar una unidad de flujo $\bar{v}$ desde el nodo origen $\bar{v}$ al nodo $K + \bar{v}$ mediante el flujo P. En la formulación (5.5)-(5.14), este flujo corresponde a un conjunto de algunas variables, en particular $x_{\bar{v}i\bar{v}} = 1$ para algunos enlaces $(\bar{v}, i) \in \mathcal{E}$, tal que $\bar{v} \in \mathcal{V}^-$, lo cual implica que $EDP^* \leq z^*_{EDP}$; que era la término al que se deseaba llegar en este párrafo. Como consecuencia de todo lo anterior, $EDP^* = z^*_{EDP}$. Dado que la formulación de ILP (5.5)-(5.14) genera un conjunto de rutas disjuntas con cardinalidad z^*_{EDP}, se puede concluir que es una formulación válida del problema EDP.

5.3.3. Implementación del modelo

Optimizador CPLEX

Para la implementación del modelo presentado en las secciones anteriores, se va a hacer uso del paquete de *software* de optimización CPLEX (https://www.ibm.com/us-en/marketplace/ibm-ilog-cplex) [121].

IBM ILOG CPLEX Optimizer recibió dicho nombre por el método simplex implementado con el lenguaje de programación C. Originalmente fue desarrollado por Robert E. Bixby y vendido comercialmente desde 1988 por CPLEX Optimization Inc. Fue adquirido por ILOG en 1997, siendo esta compañia comprada posteriormente por IBM en enero de 2009.

Para resolver los problemas de programación lineal, CPLEX implementa optimizadores basados en los algoritmos simplex (primarios y duales), así como algoritmos *barrier* logarítmicos primal-dual y de cribado (*sifting*).

CPLEX también puede resolver ciertos problemas en los que la función objetivo no es lineal sino cuadrática. Dichos problemas se conocen como programas cuadráticos o QP. Asímismo también puede dar respuesta a ciertos tipos de problemas restringidos cuadráticamente (programas QCPs cuadráticamente restringidos).

Esta herramienta también trata programas enteros mixtos o MIP debido a que puede combinar variables continuas y discretas (por ejemplo, enteros) tanto en la función objetivo como en las restricciones. Puede resolver los MIP con objetivos lineales (programas lineales enteros mixtos o MILP), los MIP con términos objetivos cuadráticos (programas cuadráticos enteros-mixtos o MIQP), y los MIP que también se encuentren restringidos cuadráticamente (programas enteros-mixtos con restricciones o MIQCP).

Además proporciona interfaces a los lenguajes de programación C++, .NET y Java.

Arquitectura de los componentes en la implementación

En este caso, se utiliza el interfaz Java que proporciona la herramienta CPLEX para realizar el modelado del programa entero. Asímismo, mediante este interfaz también se pondrán a

disposición del algoritmo metaheurístico las soluciones generadas por CPLEX, en caso de que no se haya alcanzado la solución óptima.

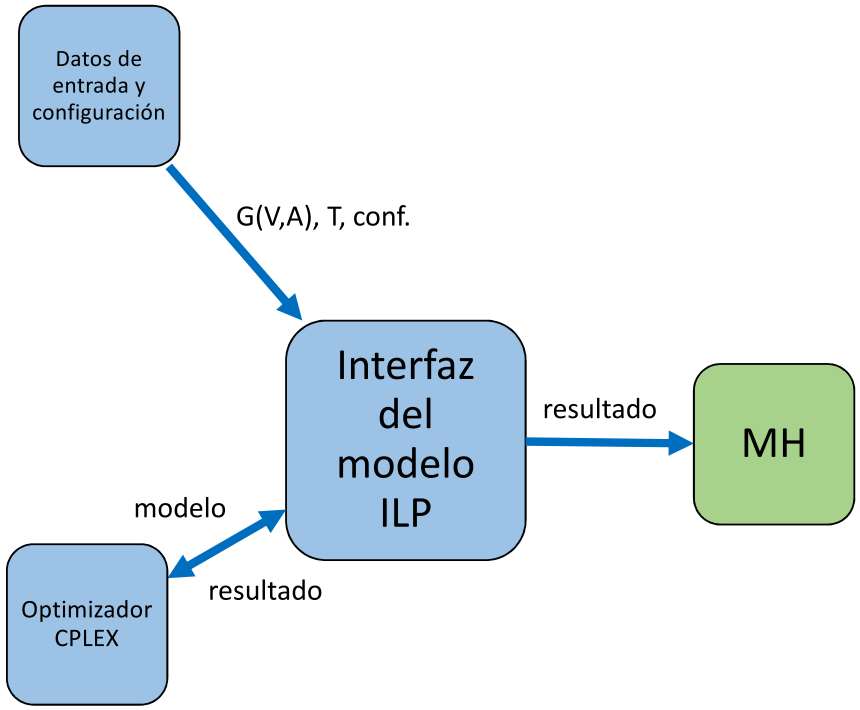

Figura 5.12: Arquitectura del modelo.

La figura 5.12 muestra la estructura de componentes necesaria para llevar a cabo la ejecución del modelo. Básicamente, en el interfaz se describe el modelo a implementar. Es decir, las variables, función objetivo y restricciones del problema. A continuación, debe tomar las instancias del ejercicio a resolver (grafos y pares terminales) así como algunos datos adicionales, como puede ser el tiempo máximo de ejecución. La información anterior se envía al optimizador CPLEX, para que resuelva la programación entera. El interfaz recoge el resultado obtenido por el paquete de optimización. En este momento, si la solución obtenida es óptima, el interfaz finaliza la ejecución. En otro caso, exporta el resultado al módulo metaheurístico.

5.4. Metaheurística poblacional

5.4.1. Introducción

En la segunda fase del proceso global participa un método metaheurístico. Este componente tomará el resultado obtenido por el modelo ILP con el objetivo de mejorarlo; es decir, proporcionar una solución que tenga asociada un mayor valor de la función objetivo.

Básicamente es un algoritmo poblacional. Pero, a diferencia de otros procedimientos, como los evolutivos [1], que se basan en la genética y en la supervivencia del mejor ("*survival of the*

[1]Los algoritmos evolutivos, o *Evolutionary Computation* (EC), se inspiran en la capacidad natural de adaptación de los seres vivos en el entorno. En el capítulo dedicado a las metaheurísticas (capítulo 3.3), y más

fittest") para mejorar los resultados; el caso que se describe en este documento se fundamenta, en ciertos aspectos, en el comportamiento cooperativo del método PSO.

Además se utilizará un algoritmo λ-*shortest paths* [2] para obtener los caminos candidatos a ser parte de la solución en el procedimiento metaheurístico.

Por último, se recuerda que los algoritmos poblacionales gestionan un conjunto de soluciones, a diferencia de los trayectoriales, que manejan únicamente un resultado [35]. El rendimiento final de estos métodos dependerá de la forma en la que se elijan y modifiquen las poblaciones.

5.4.2. El sistema multiagente

El algoritmo propuesto es un sistema multiagente. Los agentes simples o individuos de la población van construyendo las soluciones del problema de una forma iterativa. Se mueven por el espacio de búsqueda, guardan y comunican la mejor solución que han encontrado en cada iteración.

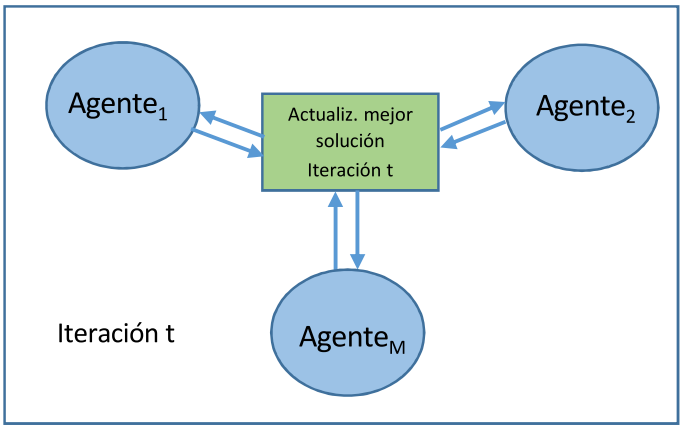

Figura 5.13: Algoritmo multiagente iterativo: estructura general.

Como se puede apreciar en las figuras 5.13 y 5.14, los agentes calculan una solución para el problema en cada repetición i, de forma que previamente necesitan disponer de la siguiente información:

- El conjunto de pares terminales.
- El conjunto de los λ-caminos más cortos asociados a cada par terminal.
- Cada agente debe tomar la solución obtenida en la iteración inmediatamente anterior.
- Cada individuo debe obtener el mejor resultado calculado por cada uno de ellos en las anteriores repeticiones.
- Asímismo, también deben tomar el mejor resultado encontrado por el total de agentes en iteraciones previas.

exactamente en la sección 3.3.5, se muestra una breve explicación de esta técnica de resolución de problemas.

[2] Para evitar problemas de nomenclatura con la variable "K" del modelo ILP, el procedimiento K-*shortest paths* será denominado como λ-*shortest paths* en esta memoria.

Y al finalizar la iteración:

- Cada agente calcula un resultado para la iteración actual.
- Cada uno de ellos actualiza la mejor solución del individuo hasta la iteración actual, si se hubiese encontrado una mejor.
- Se reemplaza el mejor resultado encontrado por todos los individuos hasta la iteración actual, si se hubiera calculado una solución mejor.

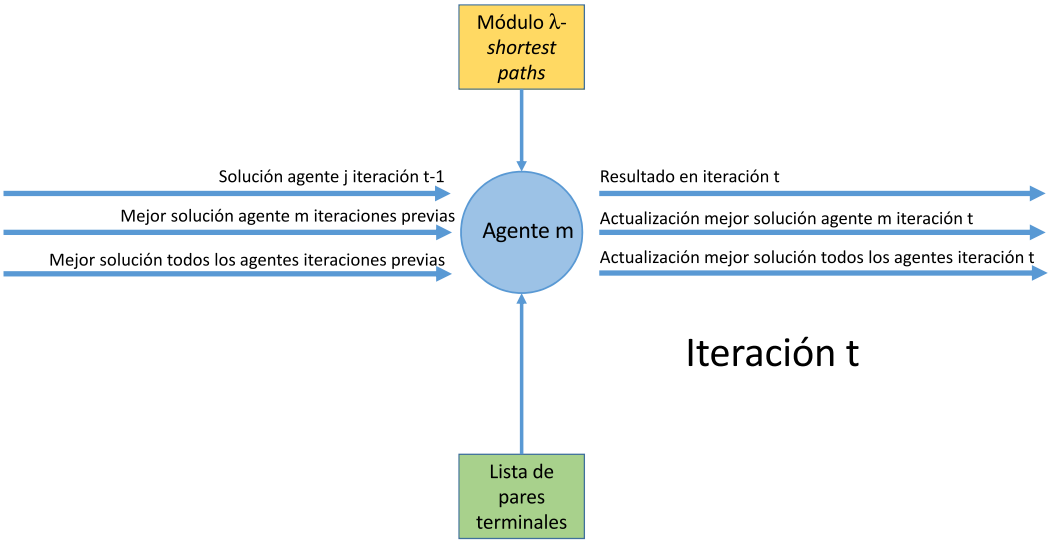

Figura 5.14: Esquema del agente en una iteración.

En los apartados siguientes se explicarán con más detalle los diferentes mecanismos planteados para la resolución del problema mediante este algoritmo.

5.4.3. Algoritmo de los λ-caminos más cortos

La obtención de los λ-caminos más cortos (λ-SP) se podría realizar mediante el algoritmo de Yen [233] (algoritmo 5.1). Se dice que este algoritmo calcula los K-caminos más cortos; pero debido a que K se refiere a los pares terminales del problema en la definición del modelo ILP, se prefiere utilizar, en este caso, otra notación que no dé lugar a equívocos.

El algoritmo de Yen utiliza dos listas A y B para crear los caminos más cortos de un par terminal determinado. La primera de ellas contiene el resultado final de las rutas más cortas; y la segunda, los caminos candidatos a convertirse en solución.

Al comienzo del proceso (ver algoritmo 5.1), se crea el recorrido más corto mediante algún procedimiento, como puede ser el algoritmo de caminos mínimos de Dijkstra [73], [175]. Una vez obtenido el camino inicial, este método se basa en la idea que indica que el camino más corto actual, por ejemplo x, $2 \leq x \leq \lambda$ puede compartir arcos y subrutas del trayecto calculado previamente $x-1$. En este proceso iterativo, se debe seleccionar un nodo del anterior camino (los que se encuentren entre la posición 1 y la posición $longitud(x-1)-1$, que actuará como nodo

Algoritmo 5.1: λ-Shortest Paths

```
Input : graph, origin, destination, λ
Output : A
{First, it is calculated the first shortest path by the Disjtra Algorithm. This route is
added into the A list. The candidate path list B is empty.}
A_0 ← dijkstraAlg(graph, origin, destination)
B ← ∅
for 2 ≤ x < λ do
   for 1 ≤ i < length(A_{x−1} − 1) do
      deviationNode ← A_{x−1}.node[i]
      {Common path from origin until the deviation node for both x-1 and x routes.}
      for each p route ∈ A do
         commonPath = p.route(0, node, i)
         if commonPath ∈ A then
            {If the common path belongs to routes previously calculated in A, it is necessary
            to avoid the node following the deviation node (the deviation node is the last
            common node).}
            graph.deleteArc(p.arc(i, i + 1))
         end if
      end for
      {The nodes which belong to the common route are eliminated, so cycles can not
      appear.}
      graph.deleteNodes(commonPath.getNodes);
      {You can now create the new path from the deviation node to destination: deviated
      route.}
      devRoute ← dijkstraAlg(graph, deviationNode, destination)
      {And the total route is created, which is added to the B list.}
      totalRoute ← commonPath ∪ devRoute
      B ← B ∪ totalRoute
      {The graph must be restored so that other deviation nodes can create new paths.}
      graph.restoreArcs(A)
      graph.restoreNodes(commonPath)
   end for
   {Sort the obtained list B and move the B's shortest one to A.}
   B ← sort(B)
   A ← A ∪ B_0
   B ← B\B_0
end for
return A
```

de desviación (*deviation node*); y seleccionar su vértice siguiente haciéndolo "incomunicable". Desde ese nodo de desviación se buscará el itinerario más corto al nodo destino del par terminal (por ejemplo, mediante el algoritmo de Dijkstra). El recorrido calculado se agregará a la ruta común formado por el trayecto que se encuentra entre el nodo fuente y el nodo de desviación. El trayecto final obtenido se incorporará a la lista B con la condición de que previamente no se encuentre en ninguna de las dos listas A o B. Después de repetir esta estrategia con todos los nodos de la ruta $x-1$ que se encuentren entre las posiciones 1 y $longitud(x-1)-1$, se toma el trayecto más corto de B para ser integrado en A.

Este proceso debe repetirse λ veces para conseguir una lista A de tamaño λ.

De manera más formal, dado un valor positivo λ, y un par terminal $k=[i_k, j_k]$ el método incluido en el procedimiento híbrido deberá obtener para dicho par terminal el conjunto de caminos más cortos definido de la siguiente manera:

$$SP(k) = \mathcal{P}_{k\Lambda} = \{p_1, ..., p_x, ..., p_\lambda\} \subseteq \mathcal{P}_k$$

de tal forma que

$SP(k) = \mathcal{P}_{k\Lambda}$ son los λ-caminos más cortos calculados para el par terminal $k=[i_k, j_k]$.

$\mathcal{P}_k$ es el conjunto de todos los caminos que conectan el par terminal k. Por tanto $\mathcal{P}_{k\Lambda}$ es un subconjunto de éste.

En cualquier caso, la creación del conjunto $\mathcal{P}_{k\Lambda}$ se realizará siguiendo los criterios incluidos en [233], [169], [168] y [167], donde:

- Los caminos $p_x \in \mathcal{P}_{k\Lambda}$ no contienen ciclos.
- Si $c(p_x)$ es la longitud del camino p_x, se cumple que $c(p_x) \leq c(p_{x+1}), \forall x \in \{1, ..., \lambda-1\}$. Es decir, que los caminos se van a calcular en orden creciente de distancia.
- $c(p_\lambda) \leq c(p), \forall p \in \mathcal{P}_k \backslash \mathcal{P}_{k\Lambda}$. Los λ-caminos contenidos en $\mathcal{P}_{k\Lambda}$ calculados para el par terminal k, tendrán menor longitud que cualquier otro itinerario que también conecte el par terminal k (incluido, por tanto en $\mathcal{P}_k$), pero no se encuentre en el conjunto $\mathcal{P}_{k\Lambda}$.

De esta manera, dado el par $[i_k, j_k] \in \mathcal{T}$, se construyen λ caminos desde i_k hasta j_k, almacenándose éstos en la lista $SP(k)$, donde k especifica el par terminal. Asumiendo que las rutas se encuentran ordenadas en orden ascendente, $SP(K) = \{SP(k,x), 1 \leq x \leq \lambda\}$.

5.4.4. Inicialización del agente

Representación de las soluciones

Uno de los aspectos clave al diseñar el algoritmo se basa en la representación del resultado. De forma específica, se define una solución como un vector con elementos K (siendo K el número de terminales), donde cada componente tiene información sobre las rutas que conexionan los distintos terminales. Más precisamente, sea p_m la solución almacenada en el agente m-ésimo; y $[i_k, j_k]$ un par de terminales en $\mathcal{T}$; entonces, si $p_{mk} = \emptyset$, indica que no se ha encontrado, en la

solución correspondiente, ninguna ruta disjunta entre i_k y j_k ; de lo contrario, $p_{mk} = \{i_k, ..., j_k\}$ representa un trayecto disjunto válido en $\mathcal{G}$.

Iteración inicial

El objetivo inicial del proceso es construir una solución factible para cada individuo o agente. De forma específica, en cada agente el algoritmo propuesto selecciona aleatoriamente uno de los pares del conjunto total $\mathcal{T}$ (por ejemplo, $[i_x, j_x]$). Entonces, los vértices de este par se conectan con el camino más corto obtenido previamente con el algoritmo de los λ-SP; es decir, $SP(x, 1)$. Esta ruta se incluye en la solución en construcción. Seguidamente a esta acción, un par diferente de terminales, $[i_y, j_y]$, también se escoge al azar. Las rutas en $SP(y)$ se examinan desde el de menor distancia, $SP(y, 1)$, hasta el de mayor longitud, $SP(y, \lambda)$, verificando si la inclusión de la ruta correspondiente produce una solución factible: es decir, sin compartir ningún enlace del grafo con el camino seleccionado previamente que conecta al par $[i_x, j_x]$). En caso de que se haya alcanzado un resultado viable, la ruta se incluiría en la solución parcial; de lo contrario, se descarta. Este método continúa buscando trayectos que conexionen siguientes pares terminales seleccionados aleatoriamente uno a uno. Este proceso finaliza cuando se hayan analizado todos los pares terminales, devolviendo la solución como un conjunto de rutas disjuntas.

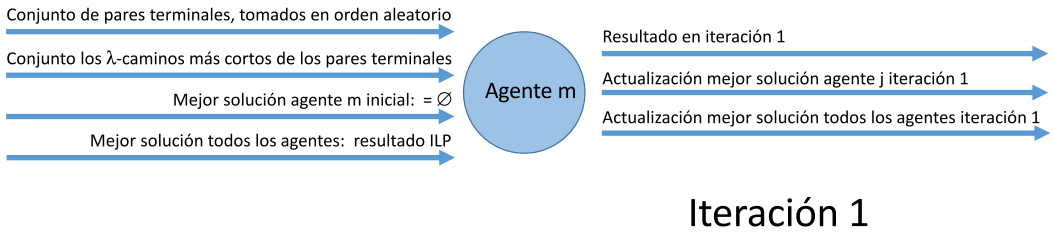

Figura 5.15: Entradas y salidas de un agente en una iteración 1.

Como se puede apreciar en la figura 5.15, los individuos del sistema, en la primera iteración, escogen de forma aleatoria los pares terminales, y los λ-caminos más cortos correspondientes a cada uno de ellos. Lógicamente, también se establece como la mejor solución alcanzada por el agente en iteraciones previas el valor Ø. Y además, se inicia el mejor resultado de todos los agentes en iteraciones anteriores con el resultado calculado con el modelo ILP, ejecutado previamente en la fase I del matheurístico.

Una vez finalizado el rastreo por cada uno de los pares, cada agente obtiene una solución inicial formada por los recorridos que conectan cada uno de las duplas (si no se obtiene un camino realizable para alguno de ellos, la solución tomará para dicha dupla el valor Ø).

Cada individuo actualizará la mejor solución obtenida hasta este momento de forma obligatoria debido a que inicialmente toma el valor Ø.

Por último, la mejor solución obtenida por todos los agentes toma el valor inicial del resultado alcanzado por el módulo ILP. Si algunos de los agentes calcula una solución que supere el valor ofrecido por dicho módulo, la mejor solución de todos los agentes será reemplazada.

5.4.5. Fase iterativa del agente

Una vez que los agentes toman una solución inicial calculada con los caminos $SP(K)$, comienza la fase iterativa. Se va a aplicar a los resultados alcanzados por los diferentes agentes una operación de cruce (*crossover*) o combinación de soluciones basada en la colaboración entre agentes. De una manera más detallada, suponiendo que el agente m calcula el resultado $p_m = \{p_1, .., p_k\}$, tal que cada elemento contiene el camino para cada par terminal entre 1 y k; se podría considerar que p_m es un cromosoma, y cada uno de los componentes del conjunto, un gen. Conociendo la operación de *crossover*, que se suele emplear en los algoritmos evolutivos sobre diferentes genes; en este caso, se va a realizar dicha acción teniendo en cuenta las soluciones generadas, tanto previamente por dicho agente, como por el resto de individuos. Más exactamente, se tendrá en cuenta, además de la solución actual del agente (se llamará resultado "inercial" del agente m), la mejor solución calculada por el individuo en repeticiones previas (resultado "cognitivo"), y el mejor resultado obtenido por todos los agentes también en las anteriores iteraciones (solución "social"). El concepto de elemento inercial, cognitivo y social se basa en los componentes del mismo nombre que se incluyen en la metaheurística PSO (ver subsección 3.3.5 del capítulo 3).

Combinación de soluciones basada en el PSO

PSO es una técnica de búsqueda poblacional [132], inspirada en el comportamiento social de agrupaciones, como pueden ser de aves o peces, que serán las partículas. La evolución de una partícula se relaciona con el cambio de posición en el espacio de búsqueda. Este método usualmente finaliza cuando la condición de detención (número de iteraciones, tiempo máximo de computación, etc.) se cumple.

Los autores del PSO describen el algoritmo de la siguiente manera: los individuos (partículas) que conviven en una sociedad tienen una "opinión" que es parte del espacio de búsqueda, compartido por todos los individuos. Cada uno de ellos puede modificar su opinión según tres factores:

- El conocimiento actual del entorno (mediante el componente inercial)
- Experiencias anteriores de la partícula o memoria del individuo (componente cognitivo).
- Experiencias anteriores de los individuos del vecindario o memoria del vecindario (componente social).

Los individuos adaptan o modifican sus opiniones a las de las partículas con más éxito de su entorno. Con el tiempo, los individuos de un entorno tendrán un conjunto de opiniones bastante relacionado.

Volviendo al algoritmo metaheurístico en desarrollo, se toman los conceptos del PSO mencionados, para llevar a cabo la etapa de combinación de soluciones en las diferentes iteraciones del proceso. Se emplea un mecanismo de cooperación mediante la operación poblacional de cruce. Este proceso se explicará con más detalle en las siguientes líneas:

Para cada agente, una vez finalizada la primera iteración, la solución inercial se consigue con el procedimiento descrito anteriormente. Teniendo en cuenta que no hay resultados precedentes,

la solución cognitiva también se establece con el mismo valor que el resultado inercial. Por otra parte, se espera que el modelo ILP logre un resultado relativamente bueno. La solución social de los individuos tomarán directamente este resultado previo. Con todo lo anterior, para cada agente se habrán establecido las soluciones inercial, cognitiva y social.

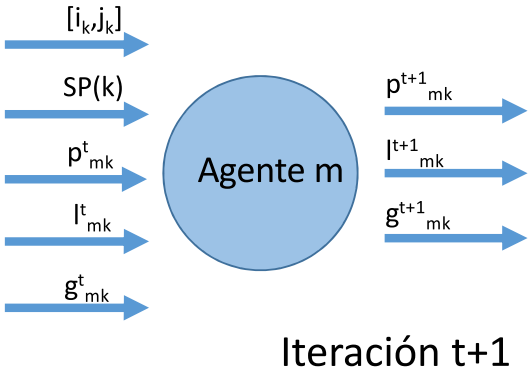

Figura 5.16: Entradas y salidas de un agente en una iteración.

$\overrightarrow{p}^t_m$, $\overrightarrow{l}^t_m$, $\overrightarrow{g}^t_m$ ($= \overrightarrow{g}^t$ debido a que es el mismo valor para todos los individuos) son los resultados almacenados por el agente m como soluciones inercial, cognitiva y social, respectivamente, en la iteración t. Suponiendo que el valor de la función objetivo (es decir, el número de rutas disjuntas que conectan los pares terminales) de estos resultados se puede calcular con el operador $|\cdot|$, se calculan los siguientes valores:

$$\psi_{iner} = \omega_{iner} \cdot |p_i^t| \tag{5.15}$$

$$\psi_{cog} = \varphi_{cog} \cdot |l_i^t| \tag{5.16}$$

$$\psi_{soc} = \varphi_{soc} \cdot |g_i^t| \tag{5.17}$$

$$\psi_{total} = \psi_{iner} + \psi_{cog} + \psi_{soc} \tag{5.18}$$

donde ω_{iner}, φ_{cog} y φ_{soc} son los coeficientes de las soluciones inercial, cognitiva y social respectivamente. Estos valores determinan la importancia de cada solución en esta etapa.

Se calcula cada solución del par terminal k del vector $\overrightarrow{p}^{t+1}_m$ (es decir, del agente m para la siguiente iteración $t+1$) de la siguiente manera:

$$p_{mk}^{t+1} = \begin{cases} p_{mk}^t & \text{if} \quad 0 \leq r < \psi_{iner} \\ l_{mk}^t & \text{if} \quad \psi_{iner} \leq r < \psi_{iner} + \psi_{cog} \\ g_{mk}^t & \text{if} \quad \psi_{iner} + \psi_{cog} \leq r \leq \psi_{total} \end{cases} \tag{5.19}$$

donde r es un número aleatorio entre 0 y ψ_{total}.

En cada iteración, cada agente toma un par terminal al azar. Por tanto, dado el agente m, y un par terminal aleatorio $[i_k, j_k] \in \mathcal{T}$, la operación de combinación de soluciones funciona de la siguiente forma:

- Si $0 \leq r < \psi_{iner}$, la dupla correspondiente mantiene el resultado previo p^t_{mk}.
- Por otra parte, si $\psi_{iner} \leq r < \psi_{iner} + \psi_{cog}$, p^{t+1}_{mk} toma el valor l^t_{mk}.

  Si además se satisface que $l^t_{mk} \neq \emptyset$ y $l^t_{mk} \neq p^t_{mk}$, éste podría ser un resultado no viable si no cumpliera con las restricciones de los enlaces disjuntos. En este caso, se deberían eliminar todas las rutas que impidiesen que la nuevo resultado fuera factible.
- De forma similar, si $\psi_{iner} + \psi_{cog} \leq r \leq \psi_{total}$, se asignaría g^t_{mk} a p^{t+1}_{mk}.

  Igual que en el caso anterior, si el nuevo resultado generase una infactibilidad en la solución, se deberían suprimir de la misma todos los caminos que imposibilitase que dicho valor fuese viable.

El mecanismo de selección del resultado actúa como una ruleta. A continuación se añade el siguiente ejemplo:

- Los datos iniciales de los pesos inercial, cognitivo y social, con los siguientes valores: $\omega_{iner} = \varphi_{cog} = \varphi_{soc} = 1/3$,
- Si los resultados del agente m son $|\overrightarrow{p}^t_m| = 10$, $|\overrightarrow{l}^t_m| = 12$, y $|\overrightarrow{g}^t_m| = 15$.

  Significa que el número de rutas disjuntas en el soluciones inercial, cognitiva y social son respectivamente 10, 12, y 15.
- El número aleatorio podrá tomar un valor entre 0 y $\psi_{total} = 10 + 12 + 15 = 37$. Al tomar el número aleatorio r:
  - Si $0 \leq r < 10$, se almacena la ruta actual que conecta el par terminal.
  - Si $10 \leq r < 22$, se establece como nueva solución del par terminal, el resultado correspondiente almacenado en la mejor solución del individuo (resultado cognitivo).
  - Finalmente, si $22 \leq r \leq 37$, la nueva ruta para la dupla $[i_k, j_k]$ sería el camino para ese mismo par terminal escogido de la solución social; es decir, del mejor resultado de todos los agentes.

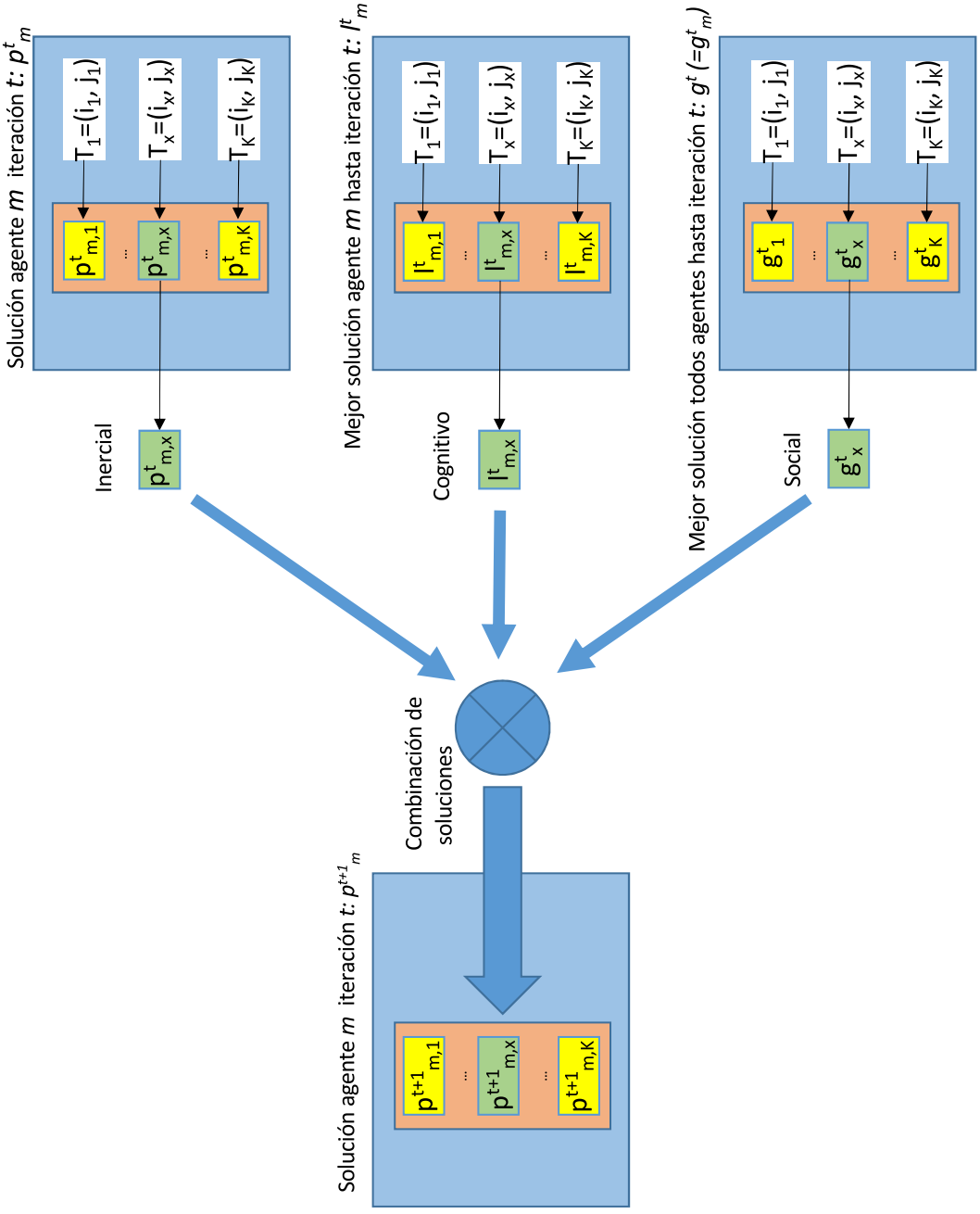

Figura 5.17: Combinación de soluciones, empleando las ecuaciones 5.15, 5.16, 5.17, 5.18 y 5.19.

En cada iteración del agente, los pares terminales se escogen al azar con el fin de favorecer la diversificación en la búsqueda. En la figura 5.16 aparece un resumen de la entrada y salida de información en el agente m para la iteración $t+1$. El individuo debe tomar la información inercial, cognitiva y social de la iteración anterior, para un par terminal escogido de forma aleatoria; y mediante la operación de cruce, se obtendrá un nuevo resultado inercial para la iteración actual. Además, se podrán reemplazar las soluciones cognitiva y social, si se obtuviese un mejor resultado respecto a las existentes.

En la figura 5.17 aparece el esquema general de la fase de combinación de soluciones del algoritmo. Tomando en la iteración $t+1$ el par terminal $x = [i_x, j_x]$, se va a aplicar las ecuaciones 5.15 hasta la 5.19, para elegir un nueva ruta, que será la p_{mx}^{t+1}. El camino resultante se habrá escogido, por tanto, mediante la combinación de soluciones inercial, cognitivo y social.

Fase de mejora

La ruta elegida en la etapa de combinación de soluciones para un agente dado va a ser siempre factible; es decir, que los arcos que componen dicho trayecto serán disjuntos. Para cumplir con este requerimiento, antes de incluir el nuevo camino en la solución del agente, se deben eliminar de esta solución las posibles rutas (si existiesen) que impidan que el nuevo camino cumpla con estos requisitos.

Y a continuación, se ejecutaría un método de postprocesamiento que verifica si los terminales no enlazados pudieran conectarse mediante alguna ruta disjunta. En esta fase de mejora o búsqueda local, se van tomando los pares terminales no conexionados, uno a uno al azar, con el objetivo de encontrar un camino disjunto que comunique cada uno de ellos. Suponiendo que se toma aleatoriamente el par $[i_q, j_q]$ sin conexión, de forma similar a la inicialización de los agentes, se comprueba la factibilidad de los λ- caminos mas cortos $SP(q)$ que conectan dicho par, y que se calcularon previamente. Se revisan las rutas del conjunto $SP(q)$, desde el de menor, $SP(q,1)$, hasta el de mayor distancia, $SP(q,\lambda)$, verificando si la inclusión de una ruta particular produce una solución factible; es decir, sin compartir ningún enlace con los trayectos ya incorporadas a la solución.

Una vez que se examinan todos los pares terminales no conectados, se debe actualizar la solución elegida en la fase de combinación (nueva solución inercial del agente) con los nuevos pares conexionados obtenidos con la búsqueda local. Además, se asignaría esta solución actualizada como nuevo valor cognitivo en caso de alcanzar un mayor número de caminos disjuntos. Igualmente ocurriría con la mejor solución de todos los individuos. En caso de que el resultado inercial de algún agente sea "mejor" que el de todos los agentes (solución social), se reeemplazaría dicho resultado global con el inercial.

Y se arranca una nueva iteración. Se vuelve a ejecutar la etapa de cruzamiento, en la que se realizará una combinación de soluciones inercial, cognitiva y social; y una fase de mejora, mediante una búsqueda local. Esta lógica se mantiene hasta alcanzar el tiempo de cálculo máximo permitido.

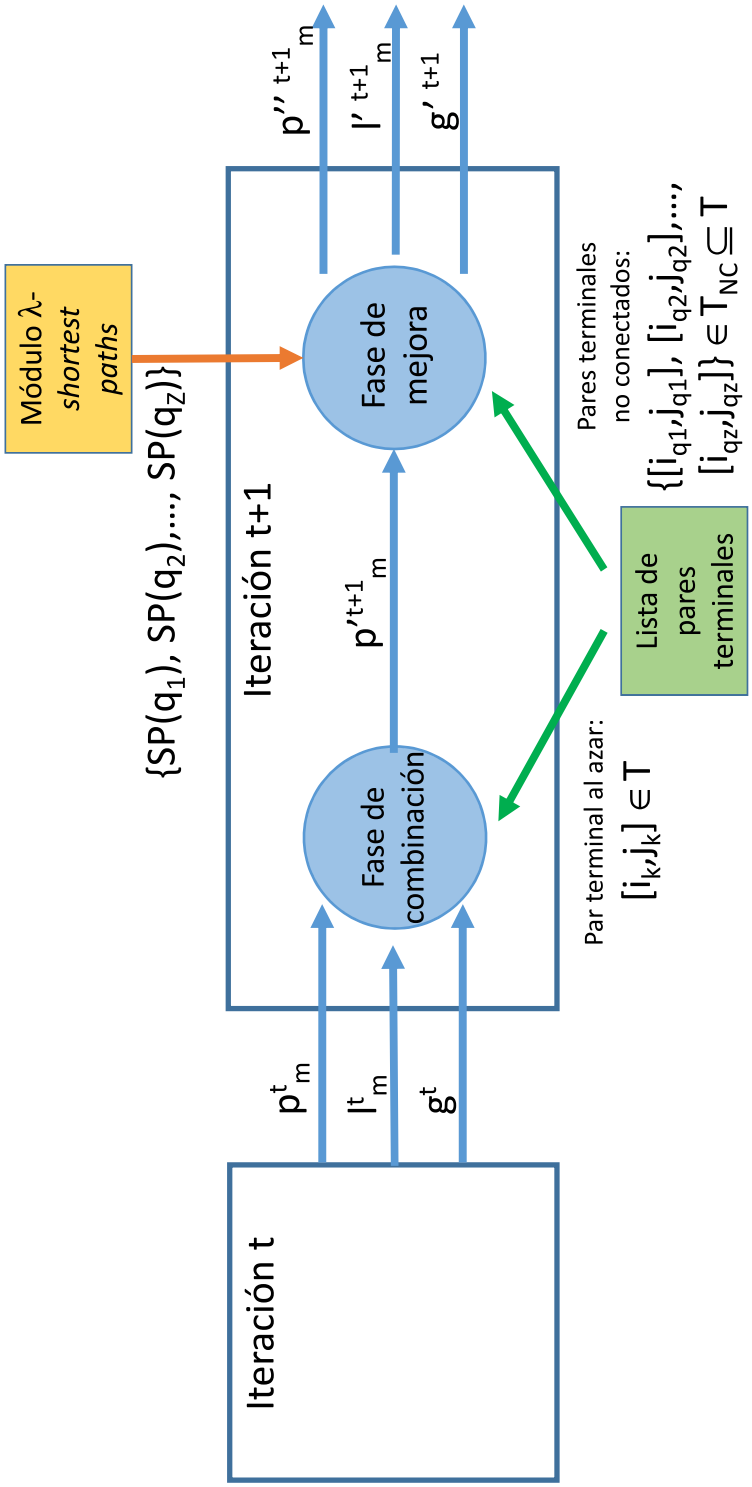

Figura 5.18: Fase iterativa para el agente m.

En la figura 5.18 aparece un esquema de la etapa iterativa. Tomando los resultados de la iteración anterior t, en la actual se va a ejecutar primeramente la fase de combinación o cruce, donde se va a obtener una nueva solución para un par terminal tomado aleatoriamente. El par escogido que se representa en la figura es $[i_k, j_k]$. Y la nueva solución para dicho par se tomará entre los siguientes: p_m^t, l_m^t y g^t. Una vez escogida la ruta y eliminados los caminos, si existiesen, que entrasen en conflicto con el nuevo trayecto, debido a que impedirían que éste cumpliese con los requerimientos de arcos disjuntos, se tiene la nueva solución $p_m^{'t+1}$. El nuevo resultado todavía puede mejorarse mediante una etapa de mejora o búsqueda local. En este caso, se tomará el conjunto de pares terminales que no se encuentren conectados en la nueva solución inercial $p_m^{'t+1}$. Estos pares sin conexionar se encuentran incluidos en $\mathcal{T}_{NC}$. Estos son $\{[i_q, j_q], [i_r, j_r], ..., [i_z, j_z]\} \in \mathcal{T}_{NC} \subseteq \mathcal{T}$.

Eligiendo de forma aleatoria cada uno de los pares terminales no conectados ($\in \mathcal{T}_{NC}$), se comprobará si algunos de los caminos SP pueden incluirse en la solución. Al finalizar el rastreo por todos los pares no conectados, se envía a la siguiente iteración $t+2$ las soluciones actualizadas: $p_m^{''t+1}$, $l_m^{'t+1}$ y $g^{'t+1}$.

Por último, señalar que el resultado final devuelto a la salida sería la solución social, común a todos los agentes.

5.5. Arquitectura general del método matheurístico

En los puntos anteriores se ha presentado en detalle las dos fases que componen el método híbrido propuesto: por un lado un modelo ILP, y por otro, un sistema metaheurístico poblacional. Para finalizar, se muestra una vista general del proceso global de todos los componentes involucrados, y las relaciones que forman parte del procedimiento matheurístico desarrollado.

En la figura 5.19 se presenta un esquema general del proceso de dos etapas. Como se puede apreciar, el modelo ILP obtiene la solución social inicial g, común para todos los agentes. Por otra parte, con la ayuda del algoritmo de los λ-caminos más cortos, los agentes puede inicializar sus soluciones inercial p y cognitiva l.

A continuación, en las diferentes iteraciones del sistema multiagente, los individuos van a ir construyendo resultados (inerciales) $p_m'^t$ (para la iteración t) por combinación de las soluciones inercial, cognitiva y social de iteraciones anteriores. Asímismo, se mejorará el resultado calculado mediante la búsqueda local, dando lugar a los valores $p_m''^t$, $l_m'^t$ y $g_m'^t$. Al finalizar las repeticiones, se devuelve el resultado común g'', que corresponde con la mejor solución encontrada.

Hay que indicar que cada una de las etapas tendrán asignadas una cantidad de tiempo determinado. De esta manera, se asegura que el sistema devolverá un resultado en un tiempo de cómputo establecido previamente.

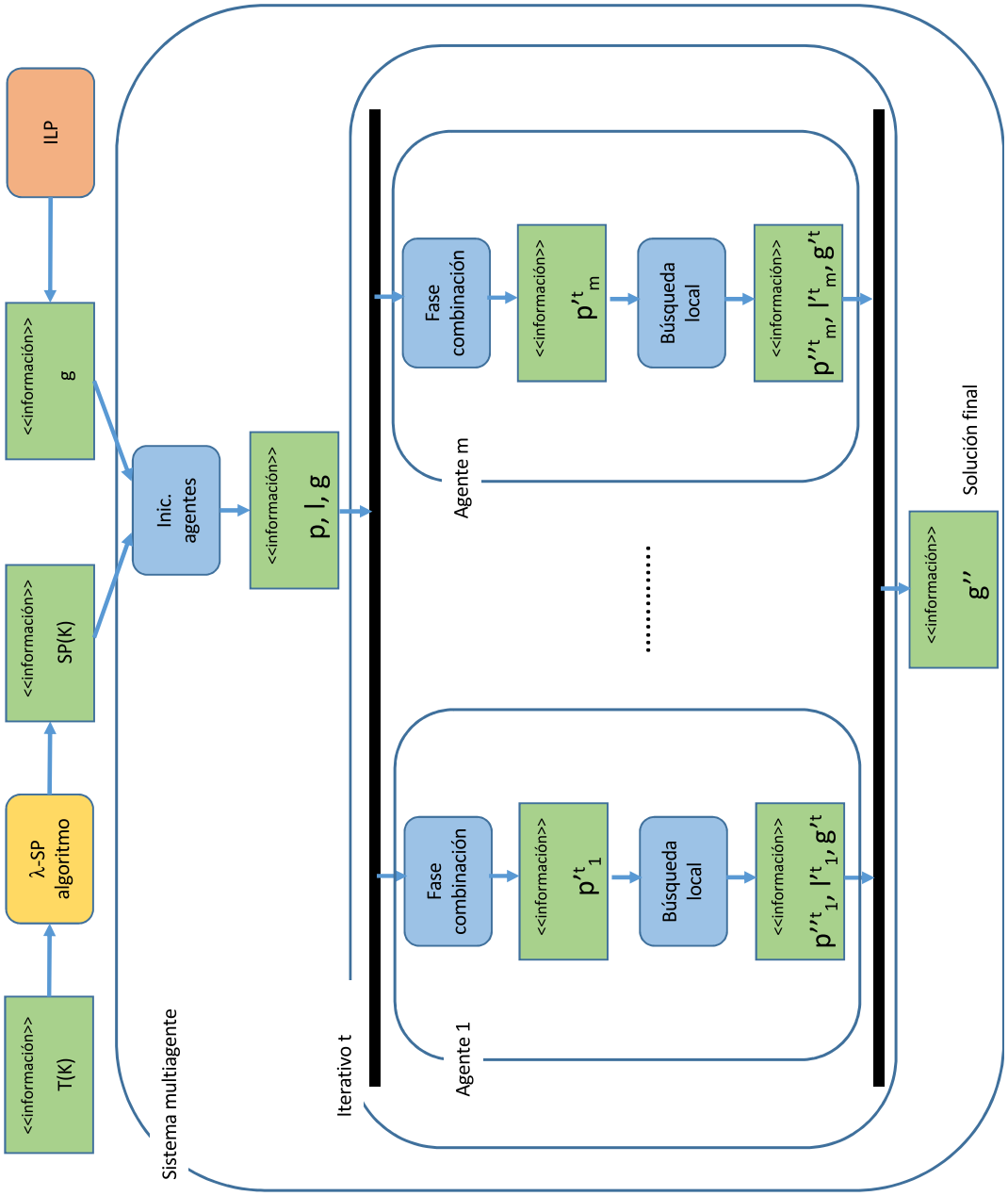

Figura 5.19: Arquitectura general del método de dos etapas ILP + sistema multiagente planteado.

5.5.1. Implementación del método metaheurístico

Por último, se incluyen los pseudocódigos del algoritmo evolutivo desarrollado. Estos son los siguientes:

- Algoritmo 5.2. Presenta el procedimiento matheurístico desarrollado en dos etapas.
- Algoritmo 5.3. Trata el metaheurístico poblacional de forma general.
- Algoritmo 5.4. Muestra la inicialización de los agentes en la primera iteración.
- Algoritmo 5.5. Enseña la etapa de cruzamiento en cada individuo como una combinación de soluciones en sucesivas iteraciones.
- Algoritmo 5.6. Por último, en la fase de mejora de soluciones, se intenta incrementar la calidad de la solución inercial en cada iteración. Se aprecia que es un proceso similar al de iniciación de individuos.

Algoritmo 5.2: `EDP: matheuristic`

Input : $\texttt{time}^{\texttt{ILP}}_{\texttt{max}}, \texttt{time}^{\texttt{MH}}_{\texttt{max}}, \lambda, \omega_{\texttt{iner}}, \varphi_{\texttt{cog}}, \varphi_{\texttt{soc}}, \texttt{indivSize}$
Output : $\texttt{g}$ {social result}

1: {start ILP Phase: ILP model calculates a initial solution}
2: $(\texttt{g}^0, \texttt{opt}) \leftarrow \texttt{solveWithCPLEX}(\texttt{time}^{\texttt{ILP}}_{\texttt{max}})$
3: **if** $(\texttt{opt} = \texttt{true})$ **then**
4: {If the solution is optimum, the procedure outputs the ILP result and ends}
5: **return** $\texttt{g}^0$
6: **else**
7: {In other case, it must start the MH Phase}
8: $\texttt{g}^{\texttt{t}} \leftarrow \texttt{metaheuristic}(\texttt{time}^{\texttt{MH}}_{\texttt{max}}, \lambda, \omega_{\texttt{iner}}, \varphi_{\texttt{cog}}, \varphi_{\texttt{soc}}, \texttt{indivSize}, \texttt{g}^0)$ {algorithm 5.3}
9: **return** $\texttt{g}^{\texttt{t}}$ {the procedure outputs the MH result and ends}
10: **end if**

Algoritmo 5.3: EDP: metaheuristic

Input : $\mathtt{time}^{\mathtt{MH}}_{\mathtt{max}}, \lambda, \omega_{\mathtt{iner}}, \varphi_{\mathtt{cog}}, \varphi_{\mathtt{soc}}, \mathtt{indivSize}, \mathtt{g}^0$
Output : $\mathtt{g}^{\mathtt{t}}$ {social result}

1: {Calculate λ-SP}
2: $\mathtt{SP}(\mathtt{T}) \leftarrow \mathtt{calculate}\lambda\mathtt{SP}(\mathtt{G}, \mathtt{T}, \lambda)$
3: {Initialize the population I: $I = \{I_1, .., I_{indivSize}\}$. A population is a set of individuals}
4: $\mathtt{I} \leftarrow \varnothing$
5: **for** $1 \leq \mathtt{m} \leq \mathtt{indivSize}$ **do**
6: $\quad \mathtt{I}^1_{\mathtt{m}} \leftarrow \mathtt{initIndividual}(\mathtt{T}, \mathtt{m}, \mathtt{SP})$ {algorithm 5.4}
7: $\quad \mathtt{I} \leftarrow \mathtt{I} \cup \{\mathtt{I}^1_{\mathtt{m}}\}$ {add this individual to the total of them}
8: **end for**
9: $\mathtt{g}^1 \leftarrow \mathtt{updateBest}(\mathtt{g}^0, \mathtt{I})$ {update social solution}
10: {Iterative process}
11: **while not** ($\mathtt{terminationCondition}(\mathtt{time}^{\mathtt{MH}}_{\mathtt{max}})$) **do**
12: $\quad$ **for** $1 \leq \mathtt{m} \leq \mathtt{indivSize}$ **do**
13: $\quad\quad$ {Combination step: crossover. Update inertial solution for each individual m}
14: $\quad\quad \mathtt{I}^{\mathtt{t}+1}_{\mathtt{m}}, \leftarrow \mathtt{combinationStep}(\mathtt{t}, \mathtt{T}, \mathtt{m}, \mathtt{SP}, \mathtt{I}^{\mathtt{t}}_{\mathtt{m}}, \mathtt{p}^{\mathtt{t}}_{\mathtt{m}}, \mathtt{l}^{\mathtt{t}}_{\mathtt{m}}, \mathtt{g}^{\mathtt{t}}, \omega_{\mathtt{iner}}, \varphi_{\mathtt{cog}}, \varphi_{\mathtt{soc}})$ {algorithm 5.5}
15: $\quad\quad$ {Local search: Update inertial and cognitive solution for each individual m}
16: $\quad\quad \mathtt{I}^{\mathtt{t}+1}_{\mathtt{m}} \leftarrow \mathtt{localSearch}(\mathtt{t}, \mathtt{T}, \mathtt{m}, \mathtt{SP}, \mathtt{p}^{\mathtt{t}+1}_{\mathtt{m}}, \mathtt{l}^{\mathtt{t}}_{\mathtt{m}})$ {algorithm 5.6}
17: $\quad$ **end for**
18: $\quad \mathtt{g}^{\mathtt{t}+1} \leftarrow \mathtt{updateBest}(\mathtt{g}^{\mathtt{t}}, \mathtt{I})$ {update social solution}
19: $\quad \mathtt{t} \leftarrow \mathtt{t} + 1$
20: **end while**
21: **return** $\mathtt{g}^{\mathtt{t}}$

Algoritmo 5.4: EDP MH: initIndividual

Input : $\mathtt{T}, \mathtt{m}, \mathtt{SP}$ {m is the individual number}
Output : $\mathtt{I}^1_{\mathtt{m}}$ {individual}

1: **while not** ($\mathtt{isFinishedSearch}(\mathtt{T})$) **do**
2: $\quad \mathtt{k} \leftarrow \mathtt{takeAtRandom}(\mathtt{T})$ {pairs are taken one by one}
3: $\quad$ {Look for a SP(k) path that meets the constraints of the problem (does not share arcs)}
4: $\quad \mathtt{p}^1_{\mathtt{m},\mathtt{k}} \leftarrow \mathtt{findSP}(\mathtt{SP}, \mathtt{k}, \mathtt{P}^1_{\mathtt{m}})$
5: $\quad$ {In this case, cognitive is equal to inertial solution}
6: $\quad \mathtt{l}^1_{\mathtt{m},\mathtt{k}} \leftarrow \mathtt{p}^1_{\mathtt{m},\mathtt{k}}$
7: $\quad \mathtt{I}'^1_{\mathtt{m}} \leftarrow \mathtt{I}^1_{\mathtt{m}} \cup \{\mathtt{p}^1_{\mathtt{m},\mathtt{k}}, \mathtt{l}^1_{\mathtt{m},\mathtt{k}}\}$
8: **end while**
9: **return** $\mathtt{I}^1_{\mathtt{m}}$

Algoritmo 5.5: `EDP MH: combinationStep`

`Input` : $\mathtt{t}, \mathtt{T}, \mathtt{m}, \mathtt{SP}, \mathtt{P}_{\mathtt{m}}^{\mathtt{t}}, \mathtt{p}_{\mathtt{m}}^{\mathtt{t}}, \mathtt{l}_{\mathtt{m}}^{\mathtt{t}}, \mathtt{g}^{\mathtt{t}}, \omega_{\mathtt{iner}}, \varphi_{\mathtt{cog}}, \varphi_{\mathtt{soc}}$
`Output` : $\mathtt{I}_{\mathtt{m}}^{\mathtt{t+1}}$ {individual m, iteration $t+1$ }

1: $\mathtt{k} \leftarrow$ `takeAtRandom`(T) {a terminal pair k is taken at random}
2: {The solution is calculated for pair k, as a combination of the current inertial, cognitive and social values}
3: $\mathtt{p}_{\mathtt{m,k}}^{\mathtt{t+1}} \leftarrow$ `crossover`$(\mathtt{p}_{\mathtt{m}}^{\mathtt{t}}, \mathtt{l}_{\mathtt{m}}^{\mathtt{t}}, \mathtt{g}^{\mathtt{t}}, \omega_{\mathtt{iner}}, \varphi_{\mathtt{cog}}, \varphi_{\mathtt{soc}})$
4: {Paths which avoid the insertion of the new inertial solution $p_{m,k}^{t+1}$ are deleted}
5: $\mathtt{I}_{\mathtt{m}}^{\mathtt{t+1}} \leftarrow$ `deletePaths`$(\mathtt{I}_{\mathtt{m}}^{\mathtt{t}}, \mathtt{p}_{\mathtt{m,k}}^{\mathtt{t+1}})$
6: **return** $\mathtt{I}_{\mathtt{m}}^{\mathtt{t+1}}$

Algoritmo 5.6: `EDP MH: localSearch`

`Input` : $\mathtt{t}, \mathtt{T}, \mathtt{m}, \mathtt{SP}, \mathtt{p}_{\mathtt{m}}^{\mathtt{t+1}}, \mathtt{l}_{\mathtt{m}}^{\mathtt{t}}$
`Output` : $\{\mathtt{p}_{\mathtt{m,TNC}}^{\mathtt{'t+1}}, \mathtt{l}_{\mathtt{m.TNC}}^{\mathtt{t+1}}\} \in \mathtt{I}_{\mathtt{m}}^{\mathtt{t+1}}$ {new inertial and cognitive solutions in individual m, iteration $t+1$}

1: $\mathtt{TNC} \leftarrow$ `getNotConnected`(T) {find terminal pairs which are not connected}
2: **while not** (`isFinishedSearch`(TNC)) **do**
3: $\mathtt{k} \leftarrow$ `takeAtRandom`(TNC) {Unconnected terminal pairs are taken one by one}
4: {Look for a SP(k) path that meets the constraints of the problem (do not share arcs)}
5: $\mathtt{p}_{\mathtt{m,k}}^{\mathtt{t+1}} \leftarrow$ `findSP`$(\mathtt{SP}, \mathtt{k}, \mathtt{P}_{\mathtt{m}}^{\mathtt{t}})$
6: {Update cognitive solution}
7: **if** $\mathtt{p}_{\mathtt{m,k}}^{\mathtt{t+1}} > \mathtt{l}_{\mathtt{m,k}}^{\mathtt{t+1}}$ **then**
8: $\mathtt{l}_{\mathtt{m,k}}^{\mathtt{t+1}} \leftarrow \mathtt{p}_{\mathtt{m,k}}^{\mathtt{t+1}}$
9: **end if**
10: {Update individual}
11: $\mathtt{I}_{\mathtt{m}}^{\mathtt{'t+1}} \leftarrow \mathtt{I}_{\mathtt{m}}^{\mathtt{t+1}} \cup (\mathtt{l}_{\mathtt{m,k}}^{\mathtt{t+1}}, \mathtt{p}_{\mathtt{m,k}}^{\mathtt{t+1}})$
12: **end while**
13: **return** $\mathtt{I}_{\mathtt{m}}^{\mathtt{t+1}}$

Capítulo 6

Resultados

6.1. Introducción

En este capítulo se van a presentar los resultados obtenidos por el método desarrollado, que será aplicado a un conjunto de grafos y pares terminales asociados. Para ello, hay que tener en cuenta los siguientes aspectos:

- Primeramente es necesario establecer el entorno de despliegue y los ejercicios que se van a realizar.
- También se debe fijar el modo en el que se efectúan las mediciones. Es decir, qué resultados se van a tomar de las pruebas para poder, posteriormente, realizar un análisis comparativo de resultados.
- Hay que indicar las características de los grafos y pares terminales asociados.
- Se deben establecer los valores que toman los diferentes parámetros asociados al algoritmo creado.
- Por último, una vez calculados los diferentes resultados, hay que llevar a cabo un estudio comparativo de métodos.

Los resultados alcanzados por el matheurístico desarrollado se contrastarán con los obtenidos por el algoritmo de colonia de hormigas y MSGA de [34], LS-SGA y LS-R de [80], el algoritmo genético (GA) de [120] y MP, MP_*reinf* y MSG de [10].

6.2. Configuración de experimentos

Esta sección introduce algunos conceptos necesarios para poder llevar a cabo la composición de las pruebas que determinen la calidad del método desarrollado. A continuación se detallan estas ideas.

6.2.1. Entorno de despliegue

Las características principales del computador y el entorno donde se realizan las mediciones son las siguientes:

- Procesador Intel Pentium Dual Core E5800 a 3,20 GHz
- Memoria RAM: 4GB
- Sistema Operativo: Windows 7.
- Java JRE 8.

Según se puede apreciar en el diagrama de contexto (figura 6.1) el método desarrollado, va a hacer uso de los siguientes elementos de entrada para poder resolver el problema:

`properties.xml`: En este archivo se indican los valores de los parámetros de configuración del algoritmo de dos etapas. Asímismo, también se muestra el conjunto de grafos y pares terminales que participarán en el ejercicio.

Entre los parámetros a configurar se encuentran los siguientes:

- Tiempo máximo de ejecución del matheurístico
- Porcentaje del tiempo de cómputo asignado a cada fase (ILP y MH).
- Número de caminos más cortos calculados para cada par terminal.
- Número de agentes (individuos) del algoritmo poblacional que participan en la resolución del problema.
- Coeficientes asociados a las soluciones inercial, cognitiva y social.

`graphName.bb` Fichero con formato BRITE [173], [172] que define el grafo. Con este formato se indica, además del número de nodos y de arcos del grafo, los arcos en sí, establecidos por el nodo inicial y el final del enlace. Asímismo, se muestra el peso del arco. En nuestro problema, todos los enlaces tendrán peso igual a 1.

`graphName.[endPairsNumber].rpairs.[endPairsNumber]`: Muestra una selección de pares terminales (o *commodities)*, indicando el nodo origen y destino de cada uno de ellos. Estos vértices se desean conectar con ayuda de los algoritmos implementados. Junto con el grafo, representan una instancia del problema.

Por otra parte, el procedimiento devuelve como salida dos ficheros, `results.txt` y `results.xlsx` con los resultados obtenidos. Estos ficheros contienen, para cada conjunto de instancias similares [1], la función objetivo medio y el tiempo de ejecución medio de la instancia.

[1] Una instancia es similar a otra si contienen el mismo grafo, y el número de pares terminales a conectar es el mismo -aunque los pares en sí sean diferentes-

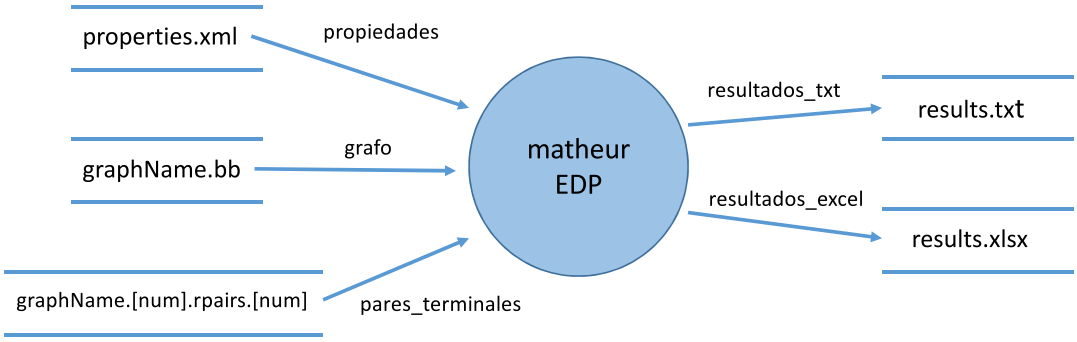

Figura 6.1: Diagrama de contexto del método.

6.2.2. Experimentos e instancias

Se define como experimento o prueba (o ensayo) a la resolución del problema para un grafo y una lista, donde cada uno de los elementos de ella contiene un conjuntos de pares terminales de similares características -es decir, con el mismo número de pares-. Debido a que una instancia es una dupla compuesta por el grafo y un conjunto de pares terminales, una prueba estará formada, por tanto, por un grupo de instancias.

Normalmente el tamaño del conjunto de pares terminales de la instancia depende del número de vértices del grafo. La razón de que se utilicen diversas instancias en la obtención de soluciones para cada experimento, es la de generalizar los resultados y que éstos no dependan de una determinada selección de los pares terminales.

Como se puede apreciar en la figura 6.2, se realiza una prueba ejecutando el algoritmos sobre las diferentes instancias asociadas al experimento. Cuando se obtienen los resultados de cada una de las instancias, se pueden obtener los resultados medios de función objetivo z_{EDP} y de tiempo de cómputo t_{med} del experimento.

Se van a medir tres parámetros en la toma de resultados de cada prueba:

- z_{EDP}: Es el valor medio de función objetivo para una instancia del experimento.
- t_{med}: Representa el tiempo medio, en segundos, de ejecución para una instancia del ensayo.

  Asímismo, en la comparación de los resultados con los métodos encontrados en el estado del arte se calculará el siguiente valor:
- $dev(\%)$: Muestra el porcentaje de desviación medio de la función objetivo del método propuesto respecto a la mejor solución encontrada en el experimento por cualquiera de los métodos que participan en la comparación. Éstos son los encontrados en el estado del arte, y los desarrollados en este trabajo.

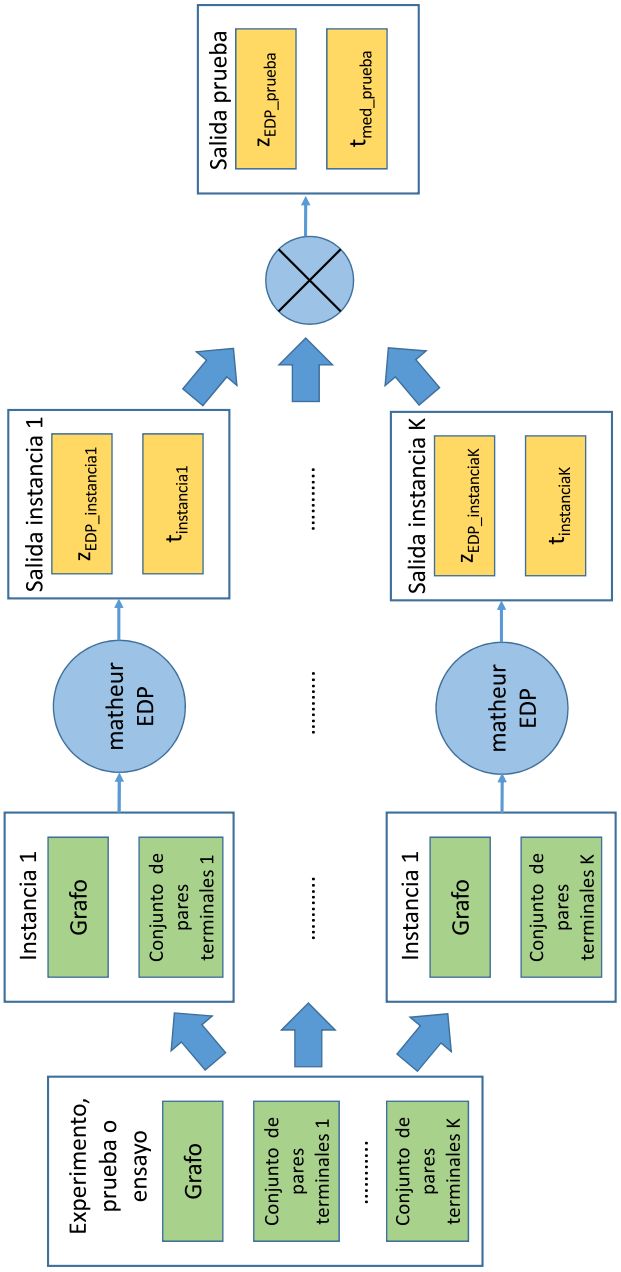

Figura 6.2: Relación entre experimentos (pruebas o ensayos) e instancias.

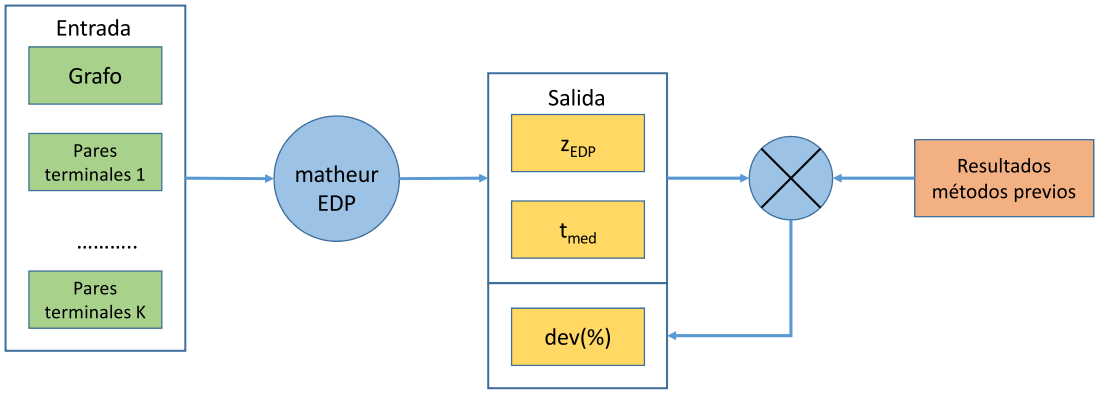

Figura 6.3: Entradas y salidas del experimento.

Como se aprecia en la figura 6.3, el método matheurístico obtiene para un grafo y el listado de conjuntos de pares terminales, el valor de función objetivo medio y en tiempo medio de cómputo para cada instancia. Como se ha comentado anteriormente, las instancias están formadas por un grafo y un conjunto de pares terminales. De esta forma, la $instancia_1 = \{grafo, pares_terminales_1\}$, la $instancia_2 = \{grafo, pares_terminales_2\}$,..., hasta la $instancia_M = \{grafo, pares_terminales_M\}$

El valor z_{EDP} junto con los resultados obtenidos para el mismo ensayo, por los métodos previos, podrán utilizarse para obtener la desviación de la función objetivo alcanzado por cada método, en el posterior análisis comparativo:

$$dev_{alg_actual} = \frac{mejor_FO_{todos_alg_comparados} - FO_{alg_actual}}{mejor_FO_{todos_alg_comparados}} \tag{6.1}$$

De la ecuación 6.1, se deduce que el cálculo de la desviación de un método para un experimento hace uso del mejor resultado obtenido de entre todos los algoritmos a examinar.

En las pruebas de [34] (en la página 15 de dicha referencia) cada experimento está formado por 20 instancias diferentes y de igual tamaño. En cambio, en [120] se utilizan grafos similares a los anteriores, pero únicamente se genera una instancia diferente por cada prueba. Por lo que, para añadir variedad a los resultados, cada ensayo está formado por la misma instancia repetida 30 veces (ver página 20 de [120]).

En [80] (ver página 25 del mismo) se realizan unos ensayos de forma similar a [120]. Únicamente se genera una instancia diferente para cada prueba. Esto hace que en cada experimento se ejecute 20 veces dicha instancia para conseguir introducir variedad en el resultado final.

En [10] (específicamente en la página 14 de la referencia), los experimentos se consideran de la misma manera que en [80]: un ensayo contiene únicamente una instancia diferente, pero se considera ejecutarla también 20 veces.

Para llevar a cabo un análisis comparativo lo más riguroso posible, los ensayos del matheurístico se realizarán de la siguiente manera:

- Para obtener las comparaciones de resultados con las alcanzadas en [34] se empleará el

conjunto de instancias (grafo, conjunto de pares terminales) utilizadas en ese estudio.

- Los resultados de [120] se comparan directamente con los obtenidos por [34], a pesar de que los pares terminales son diferentes. Como no se disponen de las instancias utilizadas en [120], el método de dos etapas se comparará con el algoritmo de [120] utilizando los experimentos llevados a cabo mediante las instancias de [34].
- Debido a que se disponen de las instancias utilizadas en [80] y [10], se llevará a cabo el análisis comparativo con estos métodos, utilizando las instancias utilizadas en cada uno de ellos. Los ensayos se efectuarán mediante la ejecución repetitiva de cada instancia hasta un valor límite igual a 20.

6.2.3. Grafos en las instancias

La información de los grafos que forman parte de las instancias ha sido tomada de las pruebas de rendimiento o *benchmarks* de [34], [80] y [10] . A continuación, se comentan la estructura de cada uno de ellos. Los grafos empleados en las pruebas se definen según los siguientes valores característicos:

- Número de vértices.
- Número de enlaces.
- Grado de adyacencia [2] medio de los nodos.

La tabla 6.1 especifica los grafos que se utilizan en los experimentos. Como se puede apreciar, existen dos grupos de grafos: Los que se utilizan en las pruebas del algoritmo de colonia de hormigas ([34]) y genéticos ([120]); y por otra parte, se emplearán otras estructuras en los experimentos comparativos con el camino óptimo restringido (COP) de [80], y la técnica de paso de mensajes MP de [10] (aunque el ACO también se puede comparar mediante el segundo conjunto de grafos).

6.2.4. Pares terminales en las instancias

Según se explicó en apartados anteriores, una instancia es un par (grafo, conjunto de pares terminales); o lo que es lo mismo, una ejecución del algoritmo con el propósito de conectar los pares terminales en el grafo. Para incluir mayor variedad en las soluciones alcanzadas, y así evitar la dependencia de los resultados al obtenido mediante un único ejemplo, se amplian las pruebas de cada grafo a un conjunto de instancias, cada una de ellas con el mismo número de pares terminales a conexionar. De esta manera se puede alcanzar resultados más generales del procedimiento, debido a que los resultados ya no dependen de una única instancia.

De igual manera, se pueden emplear un mismo grafo para diversos experimentos. En este caso, cada experimento o ensayo tendrá asociado instancias con diferentes números de pares terminales; y se obtendrán resultados dispares según el número de pares a conectar (cuyos valores se suelen corresponder a $0{,}10|V|$, $0{,}25|V|$ y $0{,}40|V|$, donde $|V|$ es el número de vértices

[2] Dos vértices i y j son adyacentes si existe una arista que los conecte. El grado de un vértice en un grafo es igual al número de nodos adyacentes a él.

Método	Grafo	$\|V\|$	$\|E\|$	Ady. media
Instancias en experimentos de [34] y [120]	graph3	164	370	4.51
	graph4	434	981	4.52
	AS-BA.R-Wax.v100e217	100	217	4.34
	bl-wr2-wht2.10.50.sdeg	500	1020	4.08
	mesh25x25	625	1200	3.84
Instancias en experimentos de [80] y [10]	bl-wr2-wht2.10-50.rand	500	1020	4.08
	bl-wr2-wht2.10.50.sdeg	500	1020	4.08
	mesh15x15	225	420	3.73
	mesh25x25	625	1200	3.84
	steinb10	75	150	4.00
	steinb16	100	200	4.00
	steinc6	500	1000	4.00
	steinc11	500	2500	10.00
	steinc16	500	12500	50.00
	planar-n50	50	135	5.4
	planar-n100	100	285	5.7
	planar-n200	200	583	5.83
	planar-n500	500	1477	5.91

Tabla 6.1: Descripción de los grafos de los *benchmarks*.

del grafo). A continuación se detallan las instancias asociadas a cada uno de los grafos que participan en las pruebas.

En la tabla 6.2 se muestran la entrada de datos que se utilizarán para realizar pruebas comparativas con los resultados obtenidos en [34] y [120]. Como se puede comprobar, cada experimento está formado por 20 instancias diferentes, cada una de ellas compuesta por el mismo número de pares terminales.

Para medir el rendimiento del matheurístico respecto a los algoritmos encontrados en [80] y [10] se van a emplear grafos y pares terminales diferentes a los presentados en la tabla 6.2. En este caso, además, se va a utilizar una única instancia (par de grafo y conjunto de pares terminales) en cada ensayo. Dicha instancia se va a ejecutar 20 veces para incluir variabilidad en los resultados promedios obtenidos por los métodos.

En la tabla 6.3 se presentan las instancias utilizadas en las experimentos de rendimiento para la familia de grafos *bl-wr2-wht2.10.50.rand* y *bl-wr2-wht2.10.50.sdeg*. Para cada grafo y número de terminales a conectar, calculados a partir de las fórmulas $0{,}10|V|$, $0{,}25|V|$ y $0{,}40|V|$, se emplean dos instancias de pares terminales adaptadas a la cantidad de conexiones a cumplimentar. Éstas son *ins1* e *ins2*. En cada ensayo, se repite la ejecución de dicha instancia 20 veces (a diferencia de las pruebas realizadas para las comparativas con [34] y [120], donde existen 20 instancias diferentes que se ejecutan una única vez cada una de ellas).

En las tablas 6.4 y 6.5 aparecen las instancias asociadas a los grafos de mallas y planos. Para los primeros se utilizan, en cada caso dos instancias (*ins1* e *ins2*) ejecutados 20 veces. Para los segundos, únicamente una instancia (*ins*).

En la tabla 6.6 aparecen las instancias que se emplearán con los grafos *steinb*. En este caso, también se servirá de una única instancia ejecutada 20 veces para la realización de cada ensayo. Del mismo modo se desarrollarán los experimentos para las instancias que incluyan los

Grafo	Núm. terminales por instancia	Núm. de instancias
graph3	16	20
	41	20
	65	20
graph4	43	20
AS-BA.R-Wax.v100e217	10	20
	25	20
	40	20
bl-wr2-wht2.10.50.sdeg	50	20
	125	20
	200	20
mesh25x25	62	20
	156	20
	250	20

Tabla 6.2: Descripción de las instancias de los experimentos para la comparativa del matheurístico y los procedimientos encontrados en [34] y [120].

Grafo	Núm. terminales por instancia	Núm. de instancias
bl-wr2-wht2.10.50.rand	50	$ins1$ repetido 20 veces
	125	$ins1$ repetido 20 veces
	200	$ins1$ repetido 20 veces
bl-wr2-wht2.10.50.rand	50	$ins2$ repetido 20 veces
	125	$ins2$ repetido 20 veces
	200	$ins2$ repetido 20 veces
bl-wr2-wht2.10.50.sdeg	50	$ins1$ repetido 20 veces
	125	$ins1$ repetido 20 veces
	200	$ins1$ repetido 20 veces
bl-wr2-wht2.10.50.sdeg	50	$ins2$ repetido 20 veces
	125	$ins2$ repetido 20 veces
	200	$ins2$ repetido 20 veces

Tabla 6.3: Descripción de las instancias de los experimentos con la familia de grafos bl-wr2-wht2.10.50 para la comparativa del matheurístico y los procedimientos de [80] y [10].

grafos *steinc* (ver tabla 6.7). Se utilizarán instancias que tratan de conectar 50, 125 y 250 pares terminales. Dichas instancias se ejecutarán 20 veces en cada prueba.

6.2.5. Configuración de los parámetros de algoritmo

Primeramente se ha de establecer un tiempo límite de ejecución del proceso. De esta manera se podrá garantizar que el método suministrará resultados competitivos con un tiempo de computación fijado previamente. De forma más exacta, el tiempo de computación máximo para la ejecución de cada instancia será la obtenida mediante la fórmula 6.2:

$$t_{max} = \rho \cdot V \cdot K \tag{6.2}$$

donde V y K son el número de vértices del grafo y de pares terminales a conectar, respectivamente. Se considera el parámetro ρ con el objetivo de alcanzar tiempos de cómputo

Grafo	Núm. terminales por instancia	Núm. de instancias
mesh15x15	22	$ins1$ repetido 20 veces
	56	$ins1$ repetido 20 veces
	90	$ins1$ repetido 20 veces
mesh15x15	22	$ins2$ repetido 20 veces
	56	$ins2$ repetido 20 veces
	90	$ins2$ repetido 20 veces
mesh25x25	62	$ins1$ repetido 20 veces
	156	$ins1$ repetido 20 veces
	250	$ins1$ repetido 20 veces
mesh25x25	62	$ins2$ repetido 20 veces
	156	$ins2$ repetido 20 veces
	250	$ins2$ repetido 20 veces

Tabla 6.4: Descripción de las instancias de los experimentos con la familia de grafos *mesh* para la comparativa del matheurístico y los procedimientos de [80] y [10].

Grafo	Núm. terminales por instancia	Núm. de instancias
planar.n50	5	ins repetido 20 veces
	12	ins repetido 20 veces
	20	ins repetido 20 veces
planar.n100	10	ins repetido 20 veces
	25	ins repetido 20 veces
	40	ins repetido 20 veces
planar.n200	20	ins repetido 20 veces
	50	ins repetido 20 veces
	80	ins repetido 20 veces
planar.n500	50	ins repetido 20 veces
	125	ins repetido 20 veces
	200	ins repetido 20 veces

Tabla 6.5: Descripción de las instancias de los experimentos con la familia de grafos planos para la comparativa del matheurístico y los procedimientos de [80] y [10].

menores a los alcanzados por los procedimientos anteriores. El valor de dicho parámetro tomará el valor $\rho = 0{,}0024$. Se ha decidido que este atributo tome este valor porque, de esta manera, se asegura que los tiempos de cómputo del algoritmo matheurístico desarrollado sean, en la mayoría de los casos, menores que los empleados por los métodos encontrados en el estado del arte, para cada uno de los experimentos.

Herramienta iRace

Con la finalidad de encontrar la configuración más apropiada para el método propuesto, se ha utilizado la herramienta iRace versión 2.2 [155]. Este *software* implementa un procedimiento iterativo capaz de encontrar la configuración de parámetros que alcanza los mejores resultados. En otras palabras, su objetivo es configurar automáticamente algoritmos de optimización mediante la búsqueda de la configuración más adecuada, empleando un conjunto de instancias de un problema de optimización. Teniendo en cuenta que cada parámetro configurable tiene asociado una distribución de muestreo, iRace se ejecuta iterativamente en tres fases:

Grafo	Núm. terminales por instancia	Núm. de instancias
steinb10	7	*ins* repetido 20 veces
	18	*ins* repetido 20 veces
	30	*ins* repetido 20 veces
steinb16	10	*ins* repetido 20 veces
	25	*ins* repetido 20 veces
	40	*ins* repetido 20 veces

Tabla 6.6: Descripción de las instancias de los experimentos con la familia de grafos *steinb* para la comparativa del matheurístico y los procedimientos de [80] y [10].

Grafo	Núm. terminales por instancia	Núm. de instancias
steinc6, steinc11, steinb16	50	*ins* repetido 20 veces
steinc6, steinc11, steinb16	125	*ins* repetido 20 veces
steinc6, steinc11, steinb16	200	*ins* repetido 20 veces

Tabla 6.7: Descripción de las instancias de los experimentos con la familia de grafos *steinc* para la comparativa del matheurístico y los procedimientos de [80] y [10].

1. Muestreo de configuraciones nuevas de acuerdo con la distribución actual: Una iteración comienza con un conjunto finito de configuraciones candidatas.

2. Selección de las mejores configuraciones: En cada paso de esta fase, las configuraciones candidatas se evalúan con una sola instancia. Después de una serie de pasos, aquellas configuraciones candidatas que obtienen peores resultados se descartan; y la etapa continúa con las configuraciones supervivientes restantes. Este procedimiento prosigue hasta alcanzar un número mínimo de configuraciones supervivientes, o un número máximo de instancias utilizadas, o una cifra máxima de experimentos ejecutados.

3. Cálculo de la distribución de las mejores configuraciones encontradas en el paso anterior: La actualización de las distribuciones consiste en modificar la media y la desviación estándar en el caso de la distribución normal; o los valores de probabilidad discretos en las distribuciones discretas.

Una vez finalizada esta iteración, que consiste en los tres pasos o etapas comentadas en las líneas anteriores, comienza la siguiente iteración: En este caso, se generan nuevas configuraciones que se añaden a las mejores configuraciones ("élites") de la iteración anterior.

El algoritmo se detiene en los siguientes casos:

1. Si se alcanza el número total de experimentos.

2. O si el número de configuraciones candidatas encontradas al inicio de una iteración no es mayor que el número de mejores configuraciones.

Por el contrario, si se alcanza el número total de iteraciones, pero todavía existen suficientes experimentos como para llevar a cabo la siguiente iteración, el número de iteraciones aumentará automáticamente y la ejecución continuará.

Parámetros

Se deben establecer valores para las siguientes variables del procedimiento:

Parámetro	Tipo	Rango	Valor por defecto
α	`Real`	[0.1,0.9]	0.5
λ	`Entero`	[10,120]	100
agentes	`Entero`	[10,150]	100
ω_{in}	`Real`	[0.1,0.9]	0.2
φ_{cogn}	`Real`	[0.1,0.9]	0.4
φ_{soc}	`Real`	[0.1,0.9]	0.4

Tabla 6.8: Configuración de parámetros.

- α. Determina el porcentaje sobre el tiempo total de ejecución del matheurístico que será utilizado en la primera fase del método; es decir, en el modelo ILP. De esta manera, si el tiempo total de cómputo es t_{max}, el tiempo máximo permitido para el proceso exacto es $\alpha \cdot t_{max}$; y el valor restante se asigna al metaheurístico poblacional $(1 - \alpha) \cdot t_{max}$.
- λ. Establece el número de caminos más cortos construido para cada par de nodos terminales.
- *agentes* (indivíduos o partículas). Es el número de agentes que participarán en la resolución del problema dentro el procedimiento poblacional.
- ω_{iner}, φ_{cog} y φ_{soc}. Son los pesos de las soluciones inercial, cognitivo y social respectivamente utilizados el algoritmo poblacional.

En la tabla 6.8 aparecen los parámetros que se deben configurar con la herramienta iRace. Asímismo, también aparece el tipo de cada variable, el rango de valores que podrían tomar, y los valores iniciales de cada uno de ellos toman.

ID	α	λ	*agent.*	ω_{iner}	φ_{cogn}	φ_{soc}
57	**0.35**	**110**	**58**	**0.33**	**0.42**	**0.25**
67	0.27	101	78	0.48	0.19	0.66
2	0.52	107	114	0.40	0.33	0.27
55	0.19	109	116	0.37	0.23	0.40

Tabla 6.9: Mejores configuraciones alcanzadas con la herramienta iRace.

La figura 6.4 representa la distribución probabilística de las diferentes variables estudiadas con la aplicación iRace. Las gráficas representan, para cada variable, la frecuencia de los valores que permiten al matheurístico calcular los mejores resultados.

Entre los parámetros representados en la figura, es necesario reseñar que *particles* se refiere a la variable *agentes*; asímismo *omega_in*, *phi_cogn* y *phi_soc* se refieren a los coeficientes ω_{iner}, φ_{cog} y φ_{soc}.

iRace también proporciona las mejores configuraciones obtenidas en el análisis de parámetros. En el caso del matheurístico, las configuraciones élites se muestran en la tabla

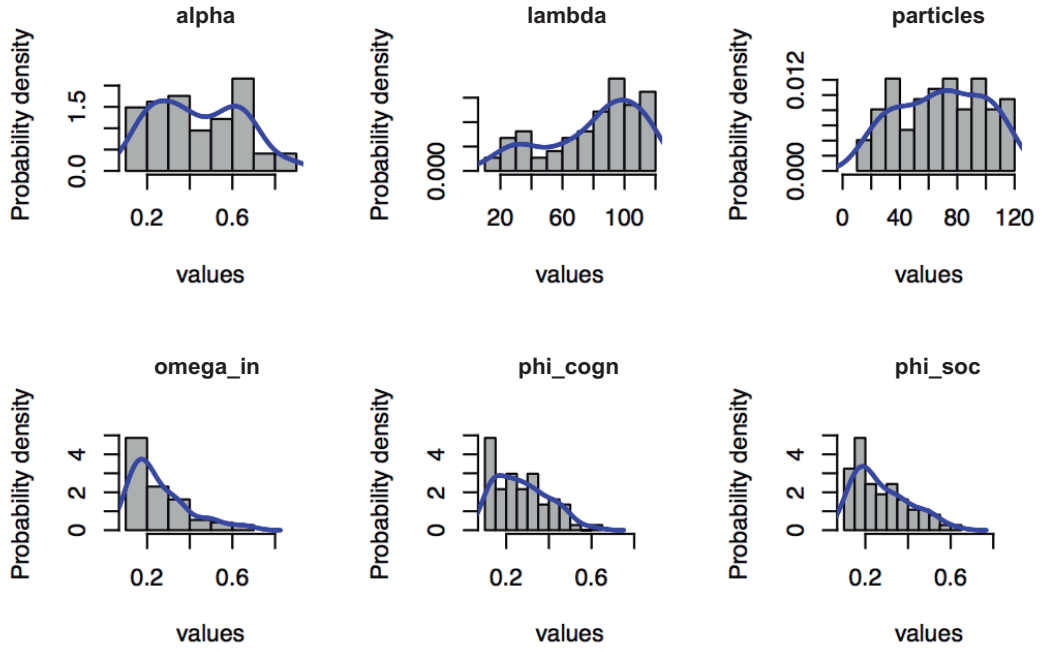

Figura 6.4: Análisis de parámetros de configuración realizado por iRace.

6.9 en orden decreciente, por lo que la mejor de entre todas ellas sería la primera; es decir la correspondiente al identificador $ID = 57$.

Con todo lo anterior, los resultados del matheurístico alcanzados en los diferentes *benchmarks* proporcionados por los diversos métodos encontrados en el estado del arte se obtendrán con la siguiente configuración:

$\alpha = 0{,}35$, $\lambda = 110$, $agentes = 58$, $\omega_{in} = 0{,}33$, $\varphi_{cogn} = 0{,}42$, y $\varphi_{soc} = 0{,}25$.

6.3. Ejecución y valoración de los experimentos

En este apartado se van a aportar los resultados alcanzados con los diversos experimentos configurados en el punto anterior. Se deberá comprobar, a partir de los valores de tiempo y función objetivo encontrados, si se cumplen con la finalidad especificada: por una parte, el matheurístico deberá alcanzar mejores resultados que cada uno de sus componentes por separado; y por otra, también deberá incrementar la función objetivo calculada por los métodos encontrados en el estado del arte, sin aumentar el tiempo de computación necesario.

6.3.1. Comparación del proceso global con cada uno de sus componentes

Se debe comprobar si la combinación del modelo exacto y el metaheurístico producen mejores resultados que la ejecución de cada componente por separado. Para ello, se muestran los resultados comparativos del modelo ILP y el algoritmo poblacional cooperativo, respecto al

método matheurístico de dos etapas. Todos ellos se ejecutan de forma aislada, y con el mismo tiempo de computación permitido.

De forma más específica, la figura 6.5 muestra un gráfico de puntos dispersos con el propósito de detallar el rendimiento del método ILP respecto al híbrido. Cada punto en el gráfico representa un experimento, y su posición viene definida por la función objetivo (número de pares conectados) obtenida por cada uno de los métodos para dicha prueba. El punto se encuentra fijado por un valor en el eje de las X, determinado por la función objetivo alcanzada por el modelo exacto en dicho ensayo; y la correspondencia en el eje de la Y, establecida por la función calculada por el matheurístico en el *test*.

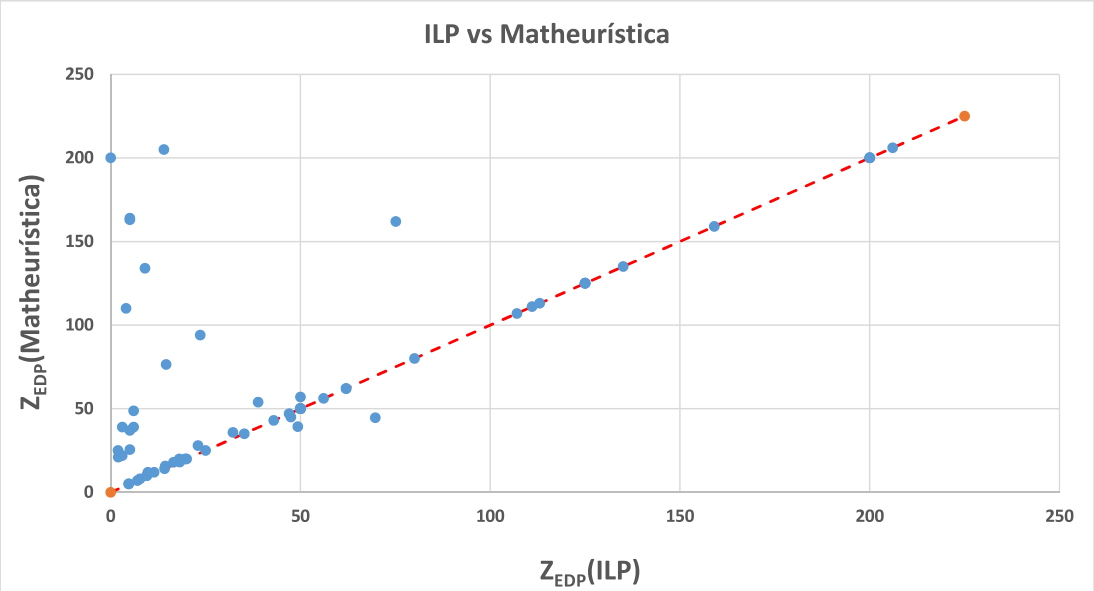

Figura 6.5: Comparación del modelo ILP con el matheurístico. En los ejes se representan el número de pares conectados por cada uno de los métodos.

Si un punto se encuentra por encima de la línea de separación, indica que para dicho ensayo el método de dos etapas alcanza mejores resultados que el modelo exacto. Si por el contrario, el punto aparece por debajo de la bisectriz, será el método exacto el que proporcione mejor rendimiento. Si se localiza sobre la línea, los dos procedimientos consiguen resultados similares de función objetivo. Como se puede apreciar en la figura, el algoritmo de dos etapas obtiene mejores valores de función objetivo que los alcanzados por el modelo exacto.

La figura 6.6 es similar a la anterior pero, en este caso se contrastarán los resultados del algoritmo poblacional multiagente frente al matheurístico. Como la mayoría de los puntos se encuentran, o bien sobre la línea, o por encima de ella, se puede deducir que el método de dos etapas también consigue mejor rendimiento que el metaheurístico.

Con todo lo anterior, se cumple uno de los propósitos del método híbrido: éste debe superar, en términos de función objetivo, a cada uno de sus componentes por separado, una vez que se define el mismo tiempo de cómputo para cada uno de ellos.

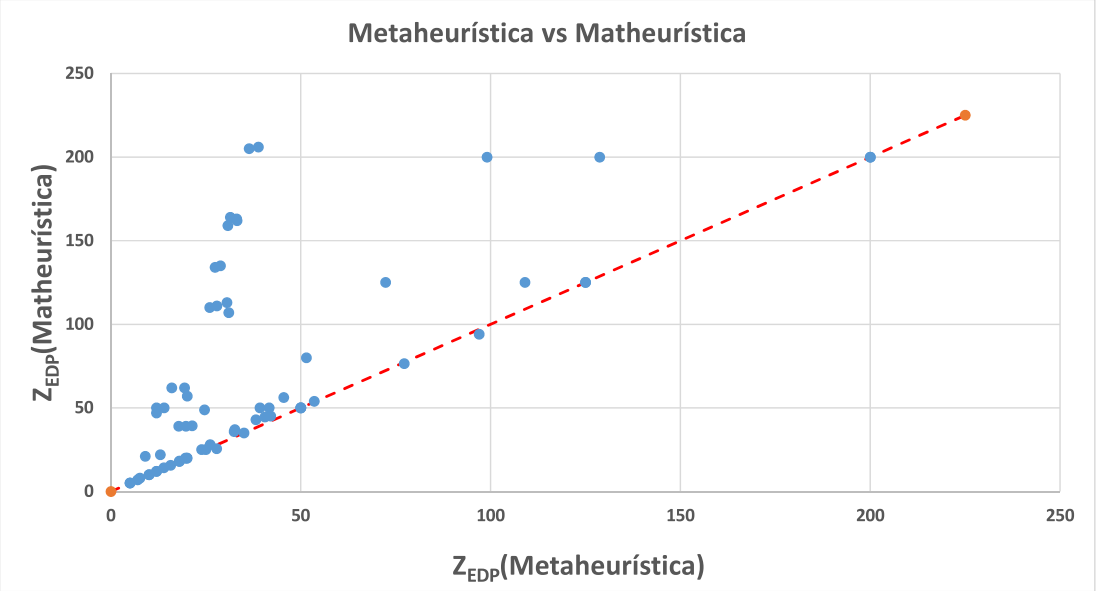

Figura 6.6: Comparación del método poblacional con el matheurístico. En los ejes se representan el número de pares conectados por cada uno de los métodos.

6.3.2. Comparación del matheurístico con los métodos del estado del arte

Una vez que se han ajustado los parámetros del algoritmo mediante el paquete de optimización iRace; y mostrado la idoneidad de la hibridación entre el procedimiento exacto y el heurístico (al obtenerse mediante la combinación resultados de mayor calidad), se debe analizar el rendimiento de la propuesta desarrollada (ILP + MH), con los mejores métodos previos identificados en el estado del arte. La comparación algorítmica es siempre una tarea compleja, ya que depende de aspectos cualitativos como el lenguaje de programación, las habilidades del desarrollador, la plataforma de *hardware*, etc. Para afrontar esta situación, se toman directamente los resultados de la fuente original logrados mediante la utilización de diferentes *benchmarks*.

Como se presenta y detalla en las tablas 6.1, 6.2, 6.3, 6.4, 6.5, 6.6 y 6.7, por un lado existen instancias similares que se emplean para las pruebas de rendimiento de los métodos ACO y MSGA de [34] y GA [120]; y por otro lado, los ensayos para los algoritmos LS-SGA y LS-R basados en el camino óptimo restringido (COP) de [80], además de MP y MP_*reinf*, que emplean la técnica de paso de mensajes MP de [10], así como el algoritmo voraz MSG también de [10] utilizan instancias similares. Por último, hay que señalar que en [80] también se incluyen pruebas comparativas del algortimo ACO desarrollado en [34].

Por lo anterior, se va a realizar la comparación del método híbrido de dos etapas (referenciado como *Math.* en las tablas de resultados), empleando los dos *benchmarks* mencionados anteriormente: con uno de ellos se realizará un análisis comparativo con los métodos MSGA, ACO y el GA; y mediante el otro grupo de instancias se realizará lo propio con los algoritmos ACO, MSGA, LS-SGA, LS-R, MP, MP_*reinf* y MSG.

También se incluyen en ambos *benchmarks* el estudio del rendimiento del modelo ILP y del metaheurístico poblacional (identificado como *Metah.* en las tablas de resultados), ejecutándose ambos de forma aislada, y con un tiempo de cómputo similar al híbrido.

La tabla 6.11 resume los resultados de los experimentos realizados con los métodos de resolución MSGA, ACO, GA, ILP, metaheurístico poblacional y matheurístico. Cada fila principal informa para cada algoritmo la siguiente información:

- z_{EDP}. Es el promedio del número de caminos disjuntos conexionados por cada instancia.
- $Dev.(\%)$. Indica la desviación porcentual promedio con respecto a la mejor solución encontrada en el experimento.
- $t(s)$. Describe el tiempo promedio de computación en segundos requerido por el procedimiento para cada instancia.
- $\#T$. Número de pares terminales en cada instancia.
- $n.inst.$ Número de instancias por experimento.

Asímismo, en el encabezado de cada columna, se indica el nombre de la familia de grafos y la cantidad de instancias de cada ensayo (o experimento).

Los resultados para los métodos MSGA, ACO y GA se toman directamente de [34] y [120]. El computador utilizado para ejecutar MSGA y ACO es un procesador Intel(R) Pentium(R) 4 a 3.06 GHz y 900 MB de memoria con un sistema operativo Linux, implementándose en C++. El GA se desarrolló en MATLAB y los experimentos se realizaron en una CPU Intel(R) i7 a 1,6 GHz y 4 GB de memoria con un sistema operativo Windows 7. Se recuerda que las pruebas de los métodos ILP, poblacional cooperativo e híbrido se realizan en una CPU Pentium(R) Dual Core E5800 a 3.20 GHz y 4 GB de memoria RAM, con el sistema operativo Windows 7 y la máquina virtual de Java JRE 8.

Para conseguir una relación entre las plataformas empleadas en las diversas pruebas de rendimiento, se va a utilizar las calificaciones de los procesadores obtenidas de la dirección web `https://www.cpubenchmark.net/` (ver tabla 6.10 [3]).

Método	Equipo	puntuación CPU
MSGA, ACO	Intel Pentium 4, 3.06 GHz	347
COP	Intel Core2 Quad Q6600, 2.40GHz	2967
GA	Intel Core i7CPU, 1.6GHz	3204
ILP, *Metah.*, *Math.*	Intel Pentium Dual-Core 3,2GHz	1910

Tabla 6.10: Cualificación de las CPUs empleadas en los experimentos.

A pesar de que la comparación de los diferentes tiempos de cómputo resulta ser una tarea compleja, se va a medir en este caso la relación de las prestaciones de las CPUs, con los tiempos de ejecución de los métodos. La plataforma experimental dedicada a los algoritmos desarrollados en esta memoria podría considerarse equivalente a la utilizada en [120]. Por otro lado, el entorno de desarrollo empleado en [34] es considerablemente más antiguo.

[3] No se ha encontrado información de la plataforma utilizada en las pruebas de los métodos MP, MP_*reinf* y MSG de [10]

Los resultados que se muestran en la tabla 6.11 indican que los tres procedimientos propuestos (ILP, poblacional cooperativo e híbrido) presentan resultados competitivos cuando se comparan con los métodos MSGA, ACO y GA. Como se esperaba, ILP presenta los mejores resultados en pequeñas y medianas instancias. Sin embargo, para entradas con mayores dimensiones (para `bl-wr2-wht2.10.50.sdeg` con 200 pares terminales a conectar, o `mesh25x25` con 156 o 250 pares de conexión), su rendimiento disminuye notablemente. Por otro lado, el poblacional desarrollado exhibe un comportamiento realmente robusto en todas las instancias, obteniendo una función objetivo promedio de 38.69 pares conectados, situándose como la segunda mejor opción; incluso destacándose sobre el método ACO, con 37.82 pares conectados de media en el *benchamark*. Finalmente, el procedimiento híbrido de dos etapas combina lo mejor de las dos estrategias algorítmicas, alcanzando el mejor rendimiento de todos los algoritmos comparados. Éste obtiene en promedio los mejores resultados en términos de función objetivo (41.88) y desviación (6.55 %).

Respecto a los tiempos de ejecución, y tomando los valores de capacidad de la CPUs que han participado en estos ensayos, se puede notar que el ratio de la capacidad computador de los experimentos de MSGA y ACO respecto a los del matheurístico es la siguiente:

$$\frac{CPU_{Math.}}{CPU_{ACO,MSGA}} = \frac{1910}{347} = 5{,}50 \tag{6.3}$$

Es decir, la CPU del matheurístico es 5.5 veces más rápido que el del MSGA o ACO.

Por otra parte, la relación entre el tiempo de cómputo del ACO y del híbrido es el siguiente:

$$\frac{t_{ACO}}{t_{Math.}} = \frac{537{,}45s}{88{,}01s} = 6{,}11 \tag{6.4}$$

que indica que el algoritmo ACO es 6 veces más lento que el híbrido.

Con los dos valores calculados, se puede apreciar que el matheurístico mejora levemente los tiempos de ejecución del ACO.

	Grafo	graph3	graph3	graph3	graph4	[1]AS-BA	[1]AS-BA	[1]AS-BA	[2]bl.sdeg	[2]bl.sdeg	[2]bl.sdeg	[3]mesh25	[3]mesh25	[3]mesh25	Media
	#T	16	41	65	43	10	25	40	50	125	200	62	156	250	
	#inst.	20	20	20	20	20	20	20	20	20	20	20	20	20	
MSGA	z_{EDP}	**15.70**	32.00	37.60	42.05	**8.05**	13.60	17.00	22.55	38.85	51.20	45.50	59.95	67.45	34.73
	Dev.(%)	0.00 %	15.01 %	13.56 %	2.10 %	0.00 %	4.23 %	21.66 %	54.21 %	44.18 %	14.67 %	19.40 %	21.47 %	27.20 %	18.33 %
	t(s)	0.96	23.24	49.27	95.74	**0.43**	4.33	9.83	208.49	534.62	1306.13	398.37	1639.32	2807.36	544.47
ACO	z_{EDP}	**15.70**	31.80	40.30	41.45	7.88	13.83	17.80	24.15	42.25	55.70	43.96	69.25	87.55	37.82
	Dev.(%)	0.00 %	15.54 %	7.36 %	3.49 %	2.11 %	2.61 %	17.97 %	50.96 %	39.30 %	7.17 %	22.13 %	9.29 %	5.50 %	14.16 %
	t(s)	**0.46**	27.95	57.90	168.87	0.16	1.82	**2.21**	48.06	190.06	798.97	679.55	1845.70	3165.17	537.45
GA	z_{EDP}	15.00	32.20	36.40	42.00	7.00	12.60	**21.70**	25.00	41.50	**60.00**	*	*	*	29.34
	Dev.(%)	5.06 %	14.48 %	16.32 %	2.21 %	13.04 %	11.27 %	0.00 %	49.24 %	40.37 %	0.00 %	*	*	*	15.20 %
	t(s)	32.00	93.00	153.00	291.00	15.00	54.00	93.00	768.00	1661.00	3592.00	*	*	*	675.20
ILP	z_{EDP}	14.35	32.20	**47.45**	**42.95**	7.75	**14.20**	18.15	**49.25**	**69.70**	38.85	56.10	14.60	23.55	33.01
	Dev.(%)	8.60 %	10.06 %	0.00 %	0.00 %	3.73 %	0.00 %	16.36 %	0.00 %	0.00 %	30.25 %	0.27 %	81.10 %	75.71 %	17.39 %
	t(s)	2.75	10.41	**18.74**	3.99	2.70	**1.67**	2.50	**24.37**	**138.82**	247.03	**41.31**	241.79	393.00	87.47
Metah.	z_{EDP}	**15.70**	32.45	42.05	38.15	7.60	13.90	18.00	21.40	40.50	53.55	45.50	**77.25**	**96.95**	38.69
	Dev.(%)	0.00 %	9.36 %	11.38 %	11.18 %	5.59 %	2.11 %	17.05 %	56.55 %	41.89 %	3.86 %	19.11 %	0.00 %	0.00 %	13.70 %
	t(s)	6.30	16.12	25.60	44.78	2.40	6.00	9.61	60.01	150.04	**240.07**	93.01	**234.13**	**375.10**	97.17
Math.	z_{EDP}	**15.70**	**35.80**	45.05	**42.95**	**8.05**	14.15	18.15	39.30	44.35	53.95	**56.25**	76.50	94.05	**41.88**
	Dev.(%)	0.00 %	0.00 %	5.06 %	0.00 %	0.00 %	0.35 %	16.36 %	20.20 %	36.08 %	3.14 %	0.00 %	0.97 %	2.99 %	6.55 %
	t(s)	3.34	11.61	22.41	**3.93**	0.84	2.15	3.01	33.17	147.63	244.33	52.29	237.45	381.94	**88.01**

Tabla 6.11: Resultados de rendimiento de los algoritmos MSGA, ACO, GA, ILP, metaheurística cooperativa y matheurística.

[1]AS-BA.R-Wax.v100e217 [2]bl-wr2-wht2.10.50.sdeg [3]mesh25x25

* No se incluyen resultados del algoritmo GA con instancias de `mesh25x25` debido a que no se incluyen en [120].

Debido a que la CPU asociada al algoritmo GA tiene prestaciones superiores al del matheurístico (3204 frente a 1910); y los tiempos de cómputo de GA son también superiores (675.20s frente a 88.01s), se deduce fácilmente que el algoritmo híbrido también mejora los tiempos del GA.

Tomando el segundo *benchmark*, en la tabla 6.12 aparecen los resultados alcanzados por los diferentes algoritmos ACO, LS-SGA, LS-R, MP, MP_*reinf*, MSG; así como los desarrollados en esta investigación: ILP, *Metah.* (metaheurística poblacional cooperativa) y *Math.* (matheurístico).

Los experimentos se agrupan por conjuntos que se encuentran asociados al mismo grafo. Por ejemplo, en la columna del grafo `bl.rand.10` se agrupan los resultados de los experimentos que emplean el grafo `bl-wr2-wht2.10.50.rand`. Para cada método, se ejecutan 6 ensayos o experimentos, cada uno de ellos con 20 instancias (debido a que aparece como número de instancias $\#inst = 6 \cdot 20$). Tres de dichos ensayos utilizan la instancia *ins1*, y los otros tres restantes *ins2* ($inst\{1, 2\}$). Por último, las instancias tratan de conectar 50, 125 y 200 pares terminales ($\#T = 50, 125, 250$). Los ensayos que se describen en estas líneas se corresponderían con las 6 primeras filas de la tabla 6.3.

Entre los resultados obtenidos y mostrados en la tabla 6.12, se aprecia que el modelo ILP desarrollado obtiene peores soluciones que en el primer *benchmark*. De hecho, en los ensayos con el grafo `steinc16` no consigue caminos de conexión. En cambio, el algoritmo multiagente *Metah.* vuelve a mostrar mejores resultados que el método exacto ($z_{EDP}^{ILP} = 41{,}60$ frente a $z_{EDP}^{Metah} = 48{,}51$). Analizando el resto de métodos del estado del arte, cabe destacar el paso de mensajes con refuerzo MP_*reinf*, que alcanza las mejores soluciones entre todos ellos (con un valor medio de $z_{EDP} = 55{,}35$ y desviación del 10.01 %). Este procedimiento supera al algoritmo poblacional multiagente, pero tiene unas prestaciones inferiores al método de dos etapas híbrido. Éste tiene asociado un valor promedio de la función objetivo igual a 60.67 y desviación de 7.23 %.

Respecto a los tiempos de cómputo, no aparece dicha información para los algoritmos de paso de mensajes (MP, MP_*reinf* y MSG). Sí que se muestran para los algoritmos LS-SGA y LS-R. Tomando el menor tiempo de ellos ($t_{LS-R} = 467{,}14s$), se aprecia que es considerablemente mayor al tiempo del matheurístico ($t_{Math} = 122{,}91s$). Analizando las prestaciones de las CPUs correspondientes, la calificación de computador dedicado a las pruebas de los algoritmos LS-SGA y LS-R es igual al 2967 (ver tabla 6.10). Un valor notablemente superior al utilizado para los ensayos del matheurístico ($CPU_{Math} = 1910$). Por tanto, se puede concluir que los tiempos de cómputo de este último procedimiento son destacadamente menores.

En ese segundo *benchmark*, el algoritmo ACO, que también aparece en la primera batería de pruebas, con $z_{EDP}^{ACO} = 50{,}27$ alcanza mejores soluciones que el poblacional cooperativo, pero lejos del método híbrido. Asímismo, los tiempos de cómputo también se acercan ($t_{ACO} = 420{,}45s$ frente a $t_{Math} = 122{,}91s$).

Por último, hay que indicar que en ambas tablas de resultados, aparecen resaltados los mayores valores de función objetivo alcanzados. Asímismo, también se muestran destacados los tiempos de cómputo mínimos encontrados, siempre que tengan asociados resultados de función objetivo competitivos.

	Grafo	[1]bl.rand	[2]bl.sdeg	[3]mesh.15	[4]mesh.25	steinb10	steinb16	steinc6	steinc11	steinc16	planar.50	planar.100	planar.200	planar.500	Media
	#T	50,125,200	50,125,200	22,56,90	62,156,250	7,18,30	10,25,40	50,125,200	50,125,200	50,125,200	5,12,20	10,25,40	20,50,80	50,125,200	
	#inst.	$6 \cdot 20$	$6 \cdot 20$	$6 \cdot 20$	$6 \cdot 20$	$3 \cdot 20$	$3 \cdot 20$	$3 \cdot 20$	$3 \cdot 20$	$3 \cdot 20$	$3 \cdot 20$	$3 \cdot 20$	$3 \cdot 20$	$3 \cdot 20$	
	inst.	$ins\{1,2\}$	$ins\{1,2\}$	$ins\{1,2\}$	$ins\{1,2\}$	ins	ins	ins	ins	ins	ins	ins	ins	ins	
ACO	z_{EDP}	30.67	29.50	29.00	48.00	17.00	23.00	87.00	124.33	**125.00**	**12.33**	23.67	38.00	66.00	50.27
	Dev.(%)	4.17 %	40.60 %	18.54 %	30.27 %	5.56 %	4.17 %	30.40 %	0.53 %	13.04 %	0.00 %	2.74 %	24.00 %	47.20 %	17.02 %
	t(s)	149.04	204.25	673.44	966.49	112.85	341.16	741.87	346.68	**23.11**	12.19	233.69	564.39	1096.63	420.45
LS-SGA	z_{EDP}	30.67	31.17	30.33	54.17	17.33	23.67	89.33	**125.00**	**125.00**	**12.33**	24.00	41.00	75.33	52.26
	Dev.(%)	4.17 %	37.25 %	14.79 %	21.31 %	3.70 %	1.39 %	28.53 %	0.00 %	13.04 %	0.00 %	1.37 %	18.00 %	39.73 %	14.10 %
	t(s)	449.89	646.14	564.79	971.15	232.54	280.78	630.24	188.74	205.39	8.98	273.92	675.61	1065.62	476.44
LS-R	z_{EDP}	30.67	29.67	31.33	57.33	**17.67**	**24.00**	94.33	**125.00**	**125.00**	**12.33**	24.00	42.00	77.33	53.13
	Dev.(%)	4.17 %	40.27 %	11.99 %	16.71 %	0.00 %	0.00 %	24.53 %	0.00 %	13.04 %	0.00 %	1.37 %	16.00 %	38.13 %	12.93 %
	t(s)	239.39	493.36	713.63	1101.05	170.53	218.86	994.67	257.94	114.42	10.98	235.09	586.72	936.23	467.14
MP	z_{EDP}	30.67	29.67	**35.60**	*	**17.67**	24.00	107.33	**125.00**	50.00	**12.33**	24.00	20.00	87.00	46.94
	Dev.(%)	4.17 %	40.27 %	0.00 %	*	0.00 %	0.00 %	14.13 %	0.00 %	65.22 %	0.00 %	1.37 %	60.00 %	30.40 %	18.12 %
	t(s)	*	*	*	*	*	*	*	*	*	*	*	*	*	*
MP	z_{EDP}	30.67	29.67	33.00	**68.83**	18.00	23.67	105.33	**125.00**	**125.00**	**12.33**	**24.33**	44.00	79.67	55.35
(reinf=	Dev.(%)	4.17 %	40.27 %	7.30 %	0.00 %	0.00 %	1.39 %	15.73 %	0.00 %	13.04 %	0.00 %	0.00 %	12.00 %	36.27 %	10.01 %
0.002)	Time(s)	*	*	*	*	*	*	*	*	*	*	*	*	*	*
MSG	z_{EDP}	29.17	26.83	27.67	50.33	17.33	23.00	90.83	**125.00**	**125.00**	**12.33**	24.00	40.83	75.00	51.33
	Dev.(%)	8.85 %	45.97 %	22.28 %	26.88 %	3.70 %	4.17 %	27.33 %	0.00 %	13.04 %	0.00 %	1.37 %	18.35 %	40.00 %	16.30 %
	Time(s)	*	*	*	*	*	*	*	*	*	*	*	*	*	*
ILP	z_{EDP}	24.17	29.50	3.83	22.33	5.33	6.67	**125.00**	**125.00**	0.00	10.67	13.33	**50.00**	**125.00**	41.60
	Dev.(%)	24.48 %	40.60 %	89.23 %	67.55 %	70.37 %	72.22 %	0.00 %	0.00 %	100.00 %	13.51 %	45.21 %	0.00 %	0.00 %	40.24 %
	t(s)	105.51	135.39	30.77	197.23	**2.78**	5.59	**13.82**	34.99	786.44	0.95	4.22	**6.85**	18.71	103.33
Metah.	z_{EDP}	25.83	25.17	19.00	28.00	17.33	23.67	97.67	**125.00**	**125.00**	**12.33**	22.33	37.33	72.00	48.51
	Dev.(%)	19.27 %	49.33 %	46.63 %	59.32 %	3.70 %	1.39 %	21.87 %	0.00 %	13.04 %	0.00 %	8.22 %	25.33 %	42.40 %	22.35 %
	t(s)	150.13	150.14	30.29	210.64	3.31	6.01	150.02	150.04	930.82	**1.49**	6.01	24.02	146.70	150.74
Math.	z_{EDP}	**32.00**	**49.67**	19.17	42.83	**17.67**	23.67	**125.00**	**125.00**	**125.00**	**12.33**	22.67	**50.00**	**125.00**	**60.67**
	Dev.(%)	0.00 %	0.00 %	46.16 %	37.77 %	0.00 %	1.39 %	0.00 %	0.00 %	0.00 %	0.00 %	6.85 %	0.00 %	0.00 %	7.23 %
	t(s)	147.15	124.69	30.67	222.86	**3.34**	**31.09**	14.00	**33.65**	958.39	**1.86**	**4.53**	6.95	**18.60**	**122.91**

Tabla 6.12: Resultados de rendimiento de los algoritmos ACO, LS-SGA, LS-R, MP, MP_*reinf*, MSG, ILP, *Metah.* y *Math.*

[1]bl-wr2-wht2.10.50.rand [2]bl-wr2-wht2.10.50.sdeg [3]mesh10x10 [4]mesh25x25

* No aparecen resultados en los experimentos presentados en [10].

6.3.3. Conclusiones de los experimentos

Como resumen de las pruebas llevadas a cabo para determinar la calidad de los algoritmos desarrollados, se destaca el uso de dos baterías de pruebas. Y en las dos, el método híbrido ha alcanzado el mejor rendimiento, obteniendo en promedio el mayor número de caminos conectados para los diferentes experimentos realizados. Los tiempos de cómputo también resultan ser menores que los empleados en los diferentes métodos del estado del arte. Con todo esto, se cumple con las expectativas de rendimiento planteadas al inicio del desarrollo del algoritmo híbrido (ver capítulo 4).

Por un lado, la colaboración entre el método exacto y el heurístico funciona bien. El algoritmo híbrido resultante alcanza mejores soluciones que cada uno de los componentes ejecutados de forma aislada. Y por otro lado, el matheurístico consigue las soluciones promedio de mayor calidad entre todos los métodos del estado del arte.

Respecto a los tiempos de ejecución, los nuevos algoritmos desarrollados emplean en la consecución de las soluciones tiempos de cómputo bastante menores que los empleados en los procedimientos encontrados en el estado del arte. Con la primera batería de pruebas, el matheurístico utiliza un tiempo $t = 88{,}01s$, mientras que el menor tiempo promedio encontrado en este primer *benchmark* es el asociado al método ACO con $t = 537{,}45s$. En el segundo grupo de experimentos, los tiempos del híbrido son también considerablemente menores $t = 122{,}91s$ frente a $t = 420{,}45s$, que es el menor tiempo empleado por los restantes métodos del estado del arte.

Por todo lo anterior, se concluye que el matheurístico desarrollado alcanza los resultados de mayor calidad con un tiempo de cómputo menor a los encontrados previamente en el estado del arte.

Capítulo 7

Conclusiones y líneas futuras de investigación

7.1. Introducción

Por medio de las investigaciones realizadas y descritas en este documento, se ha desarrollado un algoritmo matheurístico de dos etapas, formado por un modelo ILP y un algoritmo poblacional cooperativo (MH), para resolver el problema clásico EDP.

En este capítulo se trata de recopilar las conclusiones y aportaciones más importantes obtenidas con el desarrollo de este trabajo.

Asímismo, señalar que con la realización de esta investigación, quedan abiertas algunas vías interesantes en ser desarrolladas. Este capítulo finaliza exponiendo algunas de esas ideas.

7.2. Conclusiones

Para la resolución del problema de los caminos disjuntos (EDP), se ha necesitado comprender primeramente las propiedades principales del ejercicio a solucionar. Su principal característica es la pertenecencia al tipo de problemas NP.

De manera muy resumida, los problemas NP, son ejercicios complejos de resolver cuando la entrada de datos empieza a aumentar. Aplicando este concepto al ejercicio en cuestión, expresa que cuando el tamaño delgrafo o el número de pares terminales a conectar empiezan a crecer, el problema podría considerarse inabordable mediante el empleo de los técnicas exhaustivas de resolución de problemas, como pueden ser los métodos exactos. Pero existen otro tipo de herramientas que permiten alcanzar resultados cercanos al óptimo, con tiempos de cómputo competitivos para estos ejercicios: son los algoritmos metaheurísticos.

Se decidió, por tanto, crear un algoritmo híbrido de dos etapas que tuviera las ventajas y fortalezas de los métodos exactos y metaheurísticos; de tal forma que se pudiese comportar de forma eficiente para cualquier tipo de entrada. Cuando la magnitud del problema sea reducida, el componente exacto podría proporcionar una solución exacta (o cercano a ella en caso de falta de tiempo de cómputo para completar la búsqueda); y si se dispone de una dimensión

importante a la entrada del algoritmo, el elemento metaheurístico tendría la tarea de alcanzar una solución competitiva en un tiempo limitado.

La calidad del algoritmo implementado se basa, en buena medida, a la correcta coordinación y comunicación entre ambos componentes:

- Desde el punto de vista de la componente metaheurística, ILP proporciona un buen punto de partida para el algoritmo multiagente aportando, en general, soluciones ventajosas;
- y tomando como referencia el modelo exacto, el procedimiento metaheurístico suele mejorar, en tiempos ajustados, las soluciones alcanzadas por el ILP que se encuentren por debajo del óptimo.

De esta manera, ambas estrategias se consideraran complementarias, y su hibridación proporciona un método efectivo en la resolución del problema.

Para medir la calidad del procedimiento propuesto, se compararon sus resultados con los obtenidos mediante otros métodos encontrados en el estado del arte:

- Algoritmo de colonia de hormigas (ACO) y voraz multiarranque (MSGA) de [34].
- Métodos del camino óptimo restringido (COP): LS-SGA y LS-R de [80].
- El algoritmo genético (GA) de [120]
- Técnicas de paso de mensajes: MP, MP_*reinf* y el algoritmo voraz multiarranque MSG de [10].

Y para llevar a cabo dicha tarea, se utilizaron dos baterías de experimentos en la validación de resultados. Un primer *benchmark* se empleó para contrastar las soluciones del método propuesto con los alcanzados en los algoritmos ACO, MSGA y el GA. Mediante el segundo conjunto de ensayos, se examinaron las soluciones del ACO (otra vez), LS-SGA, LS-R, MP, MP_*reinf* y MSG. En ambos casos, además del matheurístico, se incluyeron los resultados del componente ILP y del algoritmo poblacional MH por separado. De esta manera, se pudo conocer en qué medida el rendimiento del método híbrido era superior al de sus dos componentes por separado.

A continuación, se van a verificar las hipótesis y objetivos planteados en este documento (capítulo 4). De esta forma, se podrá comprobar si se han cumplido con las expectativas creadas con el desarrollo del método de dos etapas ideado.

Respecto a las hipótesis sugeridas, hay que realizar las siguientes consideraciones:

- El modelo ILP alcanza buenos resultados cuando la dimensión de la instancia es reducida. Tomando la primera batería de experimentos, ILP suele obtener soluciones competitivas excepto en los casos de los grafos `bl-wr2-wht2.10.50.sdeg` con 200 pares terminales a conectar, o `mesh25x25` con 156 o 250 pares de conexión, que son las instancias con las mayores dimensiones. Cuando se escoge el segundo *benchmark*, existen experimentos como los encontrados con los grafos `mesh15x15`, `mesh25x25`, `steinb10`, `steinb16` o `steinc16` en los que los resultados del módulo exacto son pobres. Esto reafirma lo expresado en las hipótesis: ILP no funciona bien cuando la entrada de datos del problema empieza a aumentar de tamaño.

- El algoritmo MH que forma parte del matheurístico, cuando se ejecuta de forma aislada, tiene un comportamiento más homogéneo en los diferentes experimentos que se realizan con las baterías de pruebas. En el primer *benchmark* suele tener una actuación bastante aceptable, colocándose en segunda posición de la clasificación (con 38.69 pares conectados), únicamente por detrás del híbrido (con 41.88). En el segundo grupo de ensayos, el algoritmo poblacional también presenta un comportamiento uniforme (48.51 pares conectados de media), pero lejos de otros métodos de optimización, como el MP_*reinf* (con 55.35), LS-R (con 53.13) o el matheurístico, con 60.67 pares conectados.

- La tercera hipótesis ideada es la relativa a la ventaja que presenta el método híbrido. En ambos conjuntos de pruebas, este procedimiento supera los resultados de los componentes del método por separado. Incluso estableciendo un tiempo de cómputo similar tanto para el modelo ILP, como para el metaheurístico poblacional cooperativo, o el matheurístico. En el primer *benchmark*, el modelo ILP obtiene un valor de 33.01 conexiones; el metaheurístico, 38.69; y el híbrido de dos etapas, 41.88. En la segunda batería de ensayos, el modelo exacto alcanza un valor de 41.60 pares terminales conectados; el algoritmo multiagente, 48.51; y el híbrido, 60.67. Con lo que se deduce que el matheurístico es capaz de coordinar sus metodos para conseguir una mejor solución global.

Respecto a los objetivos de la investigación, se deducen las siguientes afirmaciones:

- El algoritmo implementado es original; es decir, no se ha desarrollado previamente. Realizado un análisis del estado del arte, se ha encontrado un artículo que utiliza un procedimiento PSO para resolver el problema de la enrutación dinámica y asignación de longitud de onda (Dynamic Routing and Wavelenght Assigment -RWA-) [116] . Éste es un problema parecido al EDP, pero tienen objetivos diferentes. Además, el algoritmo poblacional toma algunos conceptos del procedimiento PSO en la cooperación entre agentes; pero en ningún caso se puede considerar un algoritmo PSO. Con ésto, se cumple con uno de los objetivos de la investigación: la creación de un algoritmo singular que resuelva el problema EDP.

- Una vez contrastado los resultados obtenidos por el algoritmo híbrido y los métodos encontrados en el estado del arte, se puede concluir que el método propuesto mejora los resultados calculados por los procedimientos previos. En la primera batería de resultados, el matheurístico alcanza un resultado de 41.88 pares conectados, mientras que el mejor método del estado del arte, el ACO, alcanza un valor de 37.82. En el segundo *benchmark*, el procedimiento híbrido también consigue la mejor solución con 60.67 pares conectados; un poco más atras queda el proceso MP_*reinf*, que con 55.35, es el método previo que consigue los resultados más altos.

- Respecto a los tiempo de ejecución, el procedimiento híbrido debe calcular los resultados en un tiempo de cómputo competitivo. Aunque la tarea de comparar los tiempos de ejecución no deja de ser un trabajo complejo, debido a la diferencia de procesadores, entornos de desarrollo y lenguajes de programación, entre otros, se aprecia de forma evidente que el procedimiento de dos etapas utiliza tiempos de cómputo menores a los

empleados en el estado del arte. Mientras éste se sirve de 88.01s de promedio en la primera batería de pruebas, y 122.91s en la segunda; el método con mejores tiempos de cómputo es el procedimiento ACO con 537.45s en el primer *benchmark* y 420.45s en el segundo.

Por último, hay que destacar que en el algoritmo híbrido desarrollado en este documento, se han calculado los caminos que conectan los pares terminales mediante el módulo de los λ-caminos más cortos, en vez de crearlos paso a paso en el componente constructivo del algoritmo metaheurístico. Este mecanismo de cálculo de rutas conlleva algunas ventajas e inconvenientes:

Entre los méritos se incluyen las siguientes:

- Se separa claramente el proceso de obtención de trayectos entre pares terminales del uso de dichas rutas en el proceso de cálculo de soluciones que cumplan con las restricciones del problema EDP.
- Se permitiría la "precarga" de rutas asociadas a pares terminales que se hayan calculado previamente, lo que implica una reducción del tiempo de cómputo total. De tal forma que si se han calculado las rutas para un par terminal, y dicho par de vértices vuelve a aparecer en una instancia con posterioridad, no sería necesario volver a calcular las rutas asociadas, debido a que se ha almacenado dicha información anteriormente [1].

Entre los inconvenientes presentados se encuentran los que aparecen a continuación:

- El algoritmo multiagente no va a crear las soluciones de forma constructiva, insertando un nuevo nodo adyacente al actual en la generación de los itinerarios; en contraste, gestiona la inserción de nuevos elementos a la solución total a nivel de rutas. El camino es el componente elemental en la creación de soluciones. En otras palabras, la granularidad de la información que se va añadiendo en la fase constructiva se encuentra a nivel de trayectos, impidiendo que exista una mayor flexibilidad en la creación de resultados si se realizase insertando nodo a nodo (a menos a priori).
- En el módulo del λ-caminos más cortos se van a calcular multitud de caminos que no cumplirán con las restricciones del problema. Creando las rutas en la fase constructiva del proceso a nivel de nodos, se crearía en cada iteración, para cada agente, un único trayecto que cumpliría con dicho requerimiento.

7.3. Nuevas ideas de desarrollo

7.3.1. Problemas relacionados

El problema resuelto es el correspondiente a los caminos disjuntos *Edge-Disjoint Paths problem* (EDP), en el cual, no se pueden compartir los enlaces de un grafo no dirigido en la creación de los diferentes caminos que conectan los pares terminales. Una versión inmediata del algoritmo implementado es su adaptación a la resolución del mismo problema pero con grafos dirigidos. De esta manera, todos los enlaces tendrían un sentido definido.

[1] A pesar de la ventaja de la "precarga" de los caminos de pares terminales aparecidos en instancias previas, las soluciones obtenidas en los experimentos de las diferentes baterías de pruebas que aparecen en el capítulo 6 del presente documento, se han llevado a cabo sin aplicar dicho mecanismo.

Una transformación algo más compleja daría como resultado un procedimiento que resolviese el problema de la generación de caminos con los nodos disjuntos (*Vertex-Disjoint Connecting Paths problem* -VDP-). Sobre la versión actual, en el algoritmo poblacional se debería modificar el proceso de cálculo de la viabilidad de las caminos creados. Si actualmente se detecta que los caminos cumplan con los requerimientos de que ningún arco del grafo sea compartido por las diversas rutas calculadas; en este caso, se deberían evitar que cada nodo del grafo participe en la construcción de más de un trayecto. Por otra parte, el modelo ILP también debería ser adaptado para cumplir con esta nueva funcionalidad.

Otro punto a considerar es la longitud de los arcos del grafo. En el presente algoritmo, los enlaces son de longitud 1 en todos los casos. Pero la aplicación del problema a entornos reales implica la generalización de las distancias entre nodos, lo que acarrea la eliminación de la restricción actual relativa a la longitud de los arcos. Teniendo en cuenta que en el algoritmo multiagente, la clasificación de los caminos ordenados por longitud para cada par terminal recae en el módulo de los K (o λ)-caminos más cortos, adaptando este componente para que tenga en cuenta la longitud de los diferentes enlaces, se conseguiría resolver el problema. Más complejo parece, a priori, que el módulo ILP tenga en cuenta las distancias entre vértices.

Los enlaces del grafo tienen otra propiedad además de la longitud; y de igual manera, el valor se encuentra restringido a uno: dicho atributo es la "capacidad". Debido a esta limitación, cada enlace únicamente puede participar en un sólo camino de la solución. Sería muy interesante adaptar el problema propuesto a un ejercicio EDPwC (o *Edge-Disjoint Paths with Congestion*) donde, cada enlace e pudiese formar parte de c_e caminos, siendo c_e la capacidad del arco e. El valor c debería tomarse de cada enlace para ser incluido en el modelo ILP. Asímismo, el procedimiento multiagente debe detectar el incumplimiento de la restricciones de la capacidad de los arcos mediante dicho valor.

Otra característica que tienen los pares terminales a conexionar, y que en la versión implementada actualmente tiene el valor de uno, es la demanda de cada uno de ellos. Generalizando este parámetro a cualquier valor mayor que uno, se resolvería el problema *Multi-commodity Flow Problem*. En este caso, además de la capacidad de arcos, se debe satisfacer la demanda d_i de cada par terminal i. El objetivo es, por tanto, maximizar la suma de las demandas que pueden ser enrutadas, cumpliendo con las restricciones de capacidad de los enlaces. Para añadir esta funcionalidad al problema, debería adaptarse tanto el modelo entero como el metaheurístico poblacional.

7.3.2. Métodos de resolución

El problema EDP se ha resuelto mediante una matheurística simple de dos pasos. En la primera etapa opera un modelo ILP que trata de obtener una solución óptima. En caso de alcanzar la optimalidad en el resultado, se devuelve la solución sin necesidad de ejecutar la segunda fase. En caso contrario, un metaheurístico poblacional toma la salida proporcionada por el primer módulo, para que la segunda etapa del procedimiento mejore el resultado inicial suministrado.

El metaheurístico es un algoritmo multiagente, en la que cada elemento genera un resultado del problema teniendo en cuenta la solución inicial. De esta forma, se intenta realizar tareas

de intensificación de la solución entregada por el modelo entero; pero además se pretende aplicar una diversificación mediante la combinación de las soluciones inercial, cognitiva y social, tomándo estos conceptos del procedimiento PSO. Como se ha observado en los resultados, la combinación del ILP y MH generan soluciones finales competitivas.

Pero, para continuar y ampliar con la investigación en el desarrollo de algoritmos de optimización que resuelvan el problema EDP, se podrían desarrollar diversas propuestas algorítmicas que se integrasen en el matheurístico tanto reemplazando al primero, como sustituyendo al segundo componente del procedimiento híbrido, es decir, al actual metaheurístico.

En la primera fase del matheurístico, se podría suplantar el modelo ILP por otro mecanismo de resolución. Se podría realizar una relajación lineal del modelo actual para, a continuación, aplicar alguna técnica que generasen soluciones enteras. Se podría estudiar el uso del método de planos de corte o el *Branch & Cut*, entre otros.

Por otra parte, se podría incluir como algoritmo en la segunda fase, un metaheurístico trayectorial o poblacional que tomase la solución proporcionada por el modelo ILP como solución inicial. Debido a que las técnicas metaheurísticas incluyen multitud de procedimientos, se podrían requerir múltiples implementaciones y validaciones para conseguir una propuesta realmente interesante. A continuación se incluyen algunas de ellas:

- Dentro de los procedimientos trayectoriales, se podría utilizar la búsqueda local iterativa (entre otros). Tomaría la salida suministrada por el modelo ILP, para aplicar tanto técnicas de diversificación mediante la "perturbación", como la búsqueda local para mejorar la solución inicial.

- En el caso de emplear metaheurísticas poblacionales, la primera opción que aparece es el algoritmo genético GA. Este algoritmo sería muy parecido al actual, en el cual las diferentes poblaciones generadas se van cruzando entre ellas y mutando la solución obtenida. A diferencia del procedimiento desarrollado, en el que la generación de nuevas soluciones tiene en cuenta los mejores resultados alcanzados hasta este momento, sin perder de vista incluir "diversificación" mediante el componente inercial; parece que el GA funcionaría a priori sin una estrategia clara en la generación de nuevas soluciones. Para solventar este problema, se podría optar por añadir alguna clase de aprendizaje en la obtención de soluciones durante las sucesivas iteraciones. Estos algoritmos poblacionales que incluyen la técnica de aprendizaje son los llamados algoritmos de estimación de distribuciones (*Estimation of Distribution Algorithms* EDAs), entre los que se encuentra el *Population-Based Incremental Learning* PBIL [22], [23] y [109].

  Este procedimiento es similar a un algoritmo genético, donde cada camino (λ-SP) tiene asociado un vector de probabilidad para ser asignado a cada población (o solución). Calculando la calidad de la solución contenida en cada población, el algoritmo toma las mejores soluciones con el objetivo de "aprender" y actualizar el vector de probabilidad de cada camino en las diferentes iteraciones.

Por último, podría ser interesante adaptar el problema a su resolución mediante la creación de rutas sin intervención del módulo de los λ-caminos más cortos.

Apéndices

Apéndice A

Resultados detallados por experimentos

A.1. Introducción

En el capítulo 6 del documento relativo a los resultados alcanzados, se representaban los valores promedio de matheurístico desarrollado, con el objetivo de realizar un mejor análisis comparativo de estas soluciones respecto a las calculadas por los métodos encontrados en el estado del arte.

A diferencia de lo anterior, en este punto se van a exponer los resultados calculados por el método híbrido de dos etapas de forma específica para cada experimento o instancia, según la batería de pruebas.

A.2. Resultados

A continuación, se representan los resultados alcanzados por el procedimiento matheurístico mediante un conjunto de pruebas. Primeramente las relativas al primer *benchmark* utilizado para, seguidamente continuar con los ensayos pertenecientes a la segunda batería de pruebas.

Se recuerda que la configuración de los parámetros es la siguiente:

- $\alpha = 0{,}35$
- $\lambda = 110$
- $agentes = 58$
- $\omega_{in} = 0{,}33$
- $\varphi_{cogn} = 0{,}42$
- $\varphi_{soc} = 0{,}25$.

y el tiempo límite de cómputo del método híbrido de calcula de la siguiente forma:

$$t_{max} = \rho \cdot V \cdot K \tag{A.1}$$

donde V y K son el número de vértices del grafo y de pares terminales a conectar, respectivamente; y $\rho = 0{,}0024$.

Hay que indicar que el tiempo de ejecución real $t(s)$ puede superar el tiempo de cómputo calculado t_{max} debido a que el metaheurístico debe completar la iteración actual una vez que se haya iniciado, para todos los agentes del proceso, independientemente de que se supere en ese momento el tiempo límite.

A.2.1. Primera batería de pruebas

Para el primer *benchmark* aparecen en las tablas los valores obtenidos por cada instancia de un ensayo o experimento. Por esta razón, cada tabla representa un experimento. Las columnas de las tablas contienen tres valores:

- Instancia: Una instancia que pertenece al ensayo.
- z_{EDP}: El valor de la función objetivo para una instancia del experimento.
- $t(s)$: El tiempo de cómputo en segundos para una instancia de la prueba.
- *óptimo*: Indica si la instancia ha alcanzado el valor óptimo en la primera fase del procedimiento; es decir, en el modelo ILP.

Experimento con el grafo `graph3` y 16 pares terminales

De la tabla 6.1 se obtiene que el grafo `graph3` tiene 164 vértices. Y en el experimento cuyos resultados aparecen en la tabla A.1 el objetivo es conectar $K = 16$ pares terminales. Por tanto el tiempo total de cómputo para cada instancia es el siguiente:

$$t_{max} = 6{,}30s$$

.

Se puede apreciar que la instancia `graph3.4` tiene un tiempo de ejecución bastante más alto que el tiempo de cómputo calculado, debido a que el metaheurístico debe iterar al menos una vez en todos los agentes del proceso, independientemente de que se supere dicho tiempo de cómputo.

Por otra parte, el tiempo medio de ejecución del módulo del cálculo de los $\lambda - SP$ por cada instancia es de $t_{\lambda-SP} = 0{,}43s$.

Experimento con el grafo `graph3` y 41 pares terminales

El siguiente objetivo es conectar $K = 41$ pares terminales. El tiempo total de cómputo para cada instancia es el siguiente:

Instancia	z_{EDP}	t(s)	óptimo
graph3.1	16	1.76	sí
graph3.2	16	1.01	sí
graph3.3	16	1.42	sí
graph3.4	16	17.04	no
graph3.5	16	1.66	sí
graph3.6	16	1.14	sí
graph3.7	16	1.10	sí
graph3.8	16	6.39	no
graph3.9	15	1.62	no
graph3.10	16	1.33	sí
graph3.11	16	0.82	sí
graph3.12	14	6.48	no
graph3.13	16	0.64	sí
graph3.14	15	6.50	no
graph3.15	16	1.47	sí
graph3.16	16	1.51	sí
graph3.17	15	1.17	sí
graph3.18	16	6.38	no
graph3.19	15	6.38	no
graph3.20	16	0.96	sí
Media	15.70	3.34	

Tabla A.1: Descripción de los resultados detallados por cada instancia, y valores promedios para el experimento con el grafo `graph3` y 16 pares terminales de conexión.

$$t_{max} = 16{,}12s$$

.

Y los resultados aparecen en la tabla A.2

Instancia	z_{EDP}	t(s)	óptimo
graph3.1	33	16.78	no
graph3.2	40	7.28	sí
graph3.3	40	6.70	sí
graph3.4	30	16.37	no
graph3.5	28	16.34	no
graph3.6	37	7.86	sí
graph3.7	38	5.85	sí
graph3.8	33	16.33	no
graph3.9	41	6.35	sí
graph3.10	36	16.31	no
graph3.11	30	16.34	no
graph3.12	31	16.34	no
graph3.13	32	16.46	no
graph3.14	38	6.06	sí
graph3.15	40	7.60	sí
graph3.16	35	16.37	no
graph3.17	39	6.37	sí
graph3.18	41	7.89	sí
graph3.19	38	6.20	sí
graph3.20	36	16.34	no
Media	35.80	11.61	

Tabla A.2: Descripción de los resultados detallados por cada instancia, y valores promedios para el experimento con el grafo `graph3` y 41 pares terminales de conexión.

El tiempo promedio para cada instancia: $t_{\lambda-SP} = 1{,}03s$.

Experimento con el grafo `graph3` y 65 pares terminales

En este caso, el propósito es conexionar $K = 65$ pares terminales. El tiempo total de cómputo para cada instancia es el siguiente:

$$t_{max} = 25{,}58s$$

.

Y los resultados aparecen en la tabla A.3

El tiempo promedio para cada instancia: $t_{\lambda-SP} = 2{,}41s$.

Experimento con el grafo `graph4` y 43 pares terminales

En la tabla 6.1 aparece el grafo `graph4` con 434 vértices. La meta consiste en unir $K = 43$ pares terminales. El tiempo total de cómputo para cada instancia es el siguiente:

Instancia	z_{EDP}	t(s)	óptimo
graph3.1	38	26.47	no
graph3.2	41	26.01	no
graph3.3	42	25.95	no
graph3.4	51	12.51	sí
graph3.5	38	25.91	no
graph3.6	58	10.23	sí
graph3.7	54	10.18	sí
graph3.8	57	11.70	sí
graph3.9	39	25.96	no
graph3.10	40	25.94	no
graph3.11	45	26.14	no
graph3.12	46	25.95	no
graph3.13	40	26.19	no
graph3.14	42	25.95	no
graph3.15	43	26.19	no
graph3.16	40	25.95	no
graph3.17	48	25.93	no
graph3.18	40	25.96	no
graph3.19	57	13.24	sí
graph3.20	42	25.94	no
Media	45.05	22.41	

Tabla A.3: Descripción de los resultados detallados por cada instancia, y valores promedios para el experimento con el grafo `graph3` y 65 pares terminales de conexión.

$$t_{max} = 44{,}78s$$

.

Y los resultados se representan en la tabla A.4

Instancia	z_{EDP}	t(s)	óptimo
graph4.1	43	5.23	sí
graph4.2	43	5.18	sí
graph4.3	43	4.08	sí
graph4.4	43	3.43	sí
graph4.5	43	6.47	sí
graph4.6	43	5.35	sí
graph4.7	43	3.33	sí
graph4.8	43	3.27	sí
graph4.9	43	3.00	sí
graph4.10	43	3.80	sí
graph4.11	43	3.15	sí
graph4.12	43	3.20	sí
graph4.13	43	3.31	sí
graph4.14	43	3.04	sí
graph4.15	43	4.70	sí
graph4.16	43	3.33	sí
graph4.17	43	5.30	sí
graph4.18	43	3.27	sí
graph4.19	42	3.13	sí
graph4.20	43	2.95	sí
Media	42.95	3.93	

Tabla A.4: Descripción de los resultados detallados por cada instancia, y valores promedios para el experimento con el grafo `graph4` y 43 pares terminales de conexión.

El tiempo promedio para cada instancia: $t_{\lambda-SP} = 0s$, debido a que no se ha iniciado el módulo $\lambda - SP$ en ninguna ocasión; únicamente se ejecutaba el modelo ILP, que alcanza los valores óptimos en todos los casos.

Experimento con el grafo `AS.BA.R-Wax.v100e217` y 10 pares terminales

En la tabla 6.1 se indica que el grafo `AS.BA.R-Wax.v100e217` tiene 100 vértices. La intención de esta prueba se trata de crear rutas que comuniquen $K = 10$ pares terminales. El tiempo total de cómputo para cada instancia es el siguiente:

$$t_{max} = 2{,}4s$$

.

Y los resultados aparecen en la tabla A.5

El tiempo promedio para cada instancia: $t_{\lambda-SP} = 0{,}05s$; un valor muy pequeño debido a que únicamente se inicia el módulo $\lambda - SP$ en dos casos.

Instancia	z_{EDP}	t(s)	óptimo
AS.BA.R-Wax.v100e217.1	7	0.62	sí
AS.BA.R-Wax.v100e217.2	7	0.41	sí
AS.BA.R-Wax.v100e217.3	7	0.35	sí
AS.BA.R-Wax.v100e217.4	9	0.70	sí
AS.BA.R-Wax.v100e217.5	8	0.43	sí
AS.BA.R-Wax.v100e217.6	7	1.06	sí
AS.BA.R-Wax.v100e217.7	7	0.60	sí
AS.BA.R-Wax.v100e217.8	9	3.56	no
AS.BA.R-Wax.v100e217.9	9	0.58	sí
AS.BA.R-Wax.v100e217.10	8	0.70	sí
AS.BA.R-Wax.v100e217.11	8	2.61	no
AS.BA.R-Wax.v100e217.12	9	0.65	sí
AS.BA.R-Wax.v100e217.13	9	0.86	sí
AS.BA.R-Wax.v100e217.14	8	1.03	sí
AS.BA.R-Wax.v100e217.15	10	0.69	sí
AS.BA.R-Wax.v100e217.16	7	0.44	sí
AS.BA.R-Wax.v100e217.17	9	0.29	sí
AS.BA.R-Wax.v100e217.18	7	0.38	sí
AS.BA.R-Wax.v100e217.19	8	0.54	sí
AS.BA.R-Wax.v100e217.20	8	0.31	sí
Media	8.05	0.84	

Tabla A.5: Descripción de los resultados detallados por cada instancia, y valores promedios para el experimento con el grafo `AS.BA.R-Wax.v100e217` y 10 pares terminales de conexión.

Experimento con el grafo `AS.BA.R-Wax.v100e217` y 25 pares terminales

En el ensayo que se trata en este caso, se debe calcular caminos que conexionen $K = 25$ pares terminales. El tiempo total de cómputo para cada instancia es el siguiente:

$$t_{max} = 6{,}00s$$

.

Y los resultados aparecen en la tabla A.6

Instancia	z_{EDP}	t(s)	óptimo
AS.BA.R-Wax.v100e217.1	13	1.79	sí
AS.BA.R-Wax.v100e217.2	16	1.57	sí
AS.BA.R-Wax.v100e217.3	14	1.07	sí
AS.BA.R-Wax.v100e217.4	13	1.92	sí
AS.BA.R-Wax.v100e217.5	12	8.69	no
AS.BA.R-Wax.v100e217.6	15	9.56	no
AS.BA.R-Wax.v100e217.7	14	2.28	sí
AS.BA.R-Wax.v100e217.8	14	1.75	sí
AS.BA.R-Wax.v100e217.9	14	1.57	sí
AS.BA.R-Wax.v100e217.10	16	1.62	sí
AS.BA.R-Wax.v100e217.11	16	1.10	sí
AS.BA.R-Wax.v100e217.12	10	1.00	sí
AS.BA.R-Wax.v100e217.13	15	1.21	sí
AS.BA.R-Wax.v100e217.14	13	1.35	sí
AS.BA.R-Wax.v100e217.15	15	1.08	sí
AS.BA.R-Wax.v100e217.16	13	1.30	sí
AS.BA.R-Wax.v100e217.17	14	1.05	sí
AS.BA.R-Wax.v100e217.18	16	1.81	sí
AS.BA.R-Wax.v100e217.19	15	1.05	sí
AS.BA.R-Wax.v100e217.20	15	1.17	sí
Media	14.15	2.15	

Tabla A.6: Descripción de los resultados detallados por cada instancia, y valores promedios para el experimento con el grafo `AS.BA.R-Wax.v100e217` y 25 pares terminales de conexión.

El tiempo promedio para cada instancia: $t_{\lambda-SP} = 0{,}20s$. Un valor muy reducido debido a que únicamente se inicia el módulo $\lambda - SP$ en dos casos.

Experimento con el grafo `AS.BA.R-Wax.v100e217` y 40 pares terminales

El objetivo es conectar $K = 40$ pares terminales. El tiempo total de cómputo para cada instancia es el siguiente:

$$t_{max} = 9{,}60s$$

.

Y los resultados aparecen en la tabla A.7

Instancia	z_{EDP}	t(s)	óptimo
AS.BA.R-Wax.v100e217.1	22	2.39	sí
AS.BA.R-Wax.v100e217.2	18	2.49	sí
AS.BA.R-Wax.v100e217.3	18	2.87	sí
AS.BA.R-Wax.v100e217.4	17	14.25	no
AS.BA.R-Wax.v100e217.5	18	1.61	sí
AS.BA.R-Wax.v100e217.6	15	3.32	sí
AS.BA.R-Wax.v100e217.7	18	2.17	sí
AS.BA.R-Wax.v100e217.8	21	2.27	sí
AS.BA.R-Wax.v100e217.9	19	2.80	sí
AS.BA.R-Wax.v100e217.10	19	2.55	sí
AS.BA.R-Wax.v100e217.11	17	1.84	sí
AS.BA.R-Wax.v100e217.12	21	1.52	sí
AS.BA.R-Wax.v100e217.13	19	3.04	sí
AS.BA.R-Wax.v100e217.14	17	2.26	sí
AS.BA.R-Wax.v100e217.15	18	2.13	sí
AS.BA.R-Wax.v100e217.16	17	1.54	sí
AS.BA.R-Wax.v100e217.17	17	1.92	sí
AS.BA.R-Wax.v100e217.18	20	2.81	sí
AS.BA.R-Wax.v100e217.19	17	1.62	sí
AS.BA.R-Wax.v100e217.20	18	4.88	sí
Media	18.15	3.01	

Tabla A.7: Descripción de los resultados detallados por cada instancia, y valores promedios para el experimento con el grafo `AS.BA.R-Wax.v100e217` y 40 pares terminales de conexión.

El tiempo promedio para cada instancia es tambien $t_{\lambda-SP} = 0{,}20s$. Un valor minúsculo debido a que únicamente se inicia el módulo $\lambda - SP$ en un caso.

Experimento con el grafo `bl-wr2-wht2.10.50.sdeg` y 50 pares terminales

La tabla 6.1 señala que el grafo `bl-wr2-wht2.10.50.sdeg` dispone de 500 vértices. La intención de este ensayo es conectar $K = 50$ pares terminales. El tiempo total de cómputo para cada instancia es el siguiente:

$$t_{max} = 60{,}00s$$

.

Y los resultados aparecen en la tabla A.8

Instancia	z_{EDP}	t(s)	óptimo
bl-wr2-wht2.10.50.sdeg.1	46	19.89	sí
bl-wr2-wht2.10.50.sdeg.2	19	60.16	no
bl-wr2-wht2.10.50.sdeg.3	21	61.84	no
bl-wr2-wht2.10.50.sdeg.4	20	61.56	no
bl-wr2-wht2.10.50.sdeg.5	50	11.99	sí
bl-wr2-wht2.10.50.sdeg.6	50	27.92	sí
bl-wr2-wht2.10.50.sdeg.7	20	60.83	no
bl-wr2-wht2.10.50.sdeg.8	50	14.68	sí
bl-wr2-wht2.10.50.sdeg.9	28	60.76	no
bl-wr2-wht2.10.50.sdeg.10	50	11.26	sí
bl-wr2-wht2.10.50.sdeg.11	50	17.93	sí
bl-wr2-wht2.10.50.sdeg.12	50	16.50	sí
bl-wr2-wht2.10.50.sdeg.13	22	60.93	no
bl-wr2-wht2.10.50.sdeg.14	15	60.90	no
bl-wr2-wht2.10.50.sdeg.15	47	20.26	sí
bl-wr2-wht2.10.50.sdeg.16	50	12.52	sí
bl-wr2-wht2.10.50.sdeg.17	50	20.56	sí
bl-wr2-wht2.10.50.sdeg.18	50	20.82	sí
bl-wr2-wht2.10.50.sdeg.19	48	26.73	sí
bl-wr2-wht2.10.50.sdeg.20	50	15.37	sí
Media	39.30	33.17	

Tabla A.8: Descripción de los resultados detallados por cada instancia, y valores promedios para el experimento con el grafo `bl-wr2-wht2.10.50.sdeg` y 50 pares terminales de conexión.

El tiempo promedio para cada instancia es $t_{\lambda-SP} = 4{,}17s$.

Experimento con el grafo `bl-wr2-wht2.10.50.sdeg` y 125 pares terminales

La finalidad de esta prueba es buscar caminos que conexionen $K = 125$ pares terminales. El tiempo total de cómputo para cada instancia es el siguiente:

$$t_{max} = 150{,}00s$$

.

Y los resultados aparecen en la tabla A.9

Instancia	z_{EDP}	t(s)	óptimo
bl-wr2-wht2.10.50.sdeg.1	46	153.98	no
bl-wr2-wht2.10.50.sdeg.2	49	153.29	no
bl-wr2-wht2.10.50.sdeg.3	37	152.46	no
bl-wr2-wht2.10.50.sdeg.4	41	152.79	no
bl-wr2-wht2.10.50.sdeg.5	42	153.03	no
bl-wr2-wht2.10.50.sdeg.6	39	152.21	no
bl-wr2-wht2.10.50.sdeg.7	41	152.27	no
bl-wr2-wht2.10.50.sdeg.8	37	152.27	no
bl-wr2-wht2.10.50.sdeg.9	44	152.48	no
bl-wr2-wht2.10.50.sdeg.10	41	153.10	no
bl-wr2-wht2.10.50.sdeg.11	35	152.31	no
bl-wr2-wht2.10.50.sdeg.12	40	154.38	no
bl-wr2-wht2.10.50.sdeg.13	43	152.12	no
bl-wr2-wht2.10.50.sdeg.14	43	153.27	no
bl-wr2-wht2.10.50.sdeg.15	37	151.99	no
bl-wr2-wht2.10.50.sdeg.16	43	152.17	no
bl-wr2-wht2.10.50.sdeg.17	108	51.27	sí
bl-wr2-wht2.10.50.sdeg.18	45	152.46	no
bl-wr2-wht2.10.50.sdeg.19	43	152.49	no
bl-wr2-wht2.10.50.sdeg.20	37	152.35	no
Media	44.55	147.63	

Tabla A.9: Descripción de los resultados detallados por cada instancia, y valores promedios para el experimento con el grafo `bl-wr2-wht2.10.50.sdeg` y 125 pares terminales de conexión.

El tiempo promedio para cada instancia es $t_{\lambda-SP} = 27{,}70s$.

Experimento con el grafo `bl-wr2-wht2.10.50.sdeg` y 200 pares terminales

El propósito, en este caso, es conectar $K = 125$ pares terminales en el grafo. El tiempo total de cómputo para cada instancia es el siguiente:

$$t_{max} = 240{,}00s$$

.

Y los resultados aparecen en la tabla A.10

El tiempo promedio para cada instancia es $t_{\lambda-SP} = 50{,}52s$.

Experimento con el grafo `mesh.25x25` y 62 pares terminales

En la tabla 6.1 se observa que el grafo `mesh.25x25` tiene 625 vértices. Mediante esta estructura, se desea obtener caminos que enlacen los $K = 62$ pares terminales. El tiempo total de cómputo para cada instancia es el siguiente:

$$t_{max} = 93{,}00s$$

Instancia	z_{EDP}	t(s)	óptimo
bl-wr2-wht2.10.50.sdeg.1	61	245.79	no
bl-wr2-wht2.10.50.sdeg.2	46	244.78	no
bl-wr2-wht2.10.50.sdeg.3	46	243.94	no
bl-wr2-wht2.10.50.sdeg.4	51	244.20	no
bl-wr2-wht2.10.50.sdeg.5	63	243.55	no
bl-wr2-wht2.10.50.sdeg.6	51	244.26	no
bl-wr2-wht2.10.50.sdeg.7	58	245.18	no
bl-wr2-wht2.10.50.sdeg.8	53	244.85	no
bl-wr2-wht2.10.50.sdeg.9	51	244.26	no
bl-wr2-wht2.10.50.sdeg.10	63	244.05	no
bl-wr2-wht2.10.50.sdeg.11	50	244.18	no
bl-wr2-wht2.10.50.sdeg.12	66	243.78	no
bl-wr2-wht2.10.50.sdeg.13	46	245.26	no
bl-wr2-wht2.10.50.sdeg.14	44	244.01	no
bl-wr2-wht2.10.50.sdeg.15	46	244.17	no
bl-wr2-wht2.10.50.sdeg.16	50	243.60	no
bl-wr2-wht2.10.50.sdeg.17	62	245.14	no
bl-wr2-wht2.10.50.sdeg.18	56	243.98	no
bl-wr2-wht2.10.50.sdeg.19	55	244.02	no
bl-wr2-wht2.10.50.sdeg.20	61	243.67	no
Media	53.95	147.63	

Tabla A.10: Descripción de los resultados detallados por cada instancia, y valores promedios para el experimento con el grafo `bl-wr2-wht2.10.50.sdeg` y 200 pares terminales de conexión.

.

Y los resultados aparecen en la tabla A.11

Instancia	z_{EDP}	t(s)	óptimo
mesh.25x25.1	62	28.19	sí
mesh.25x25.2	62	37.07	sí
mesh.25x25.3	47	97.29	no
mesh.25x25.4	46	20.06	sí
mesh.25x25.5	62	94.29	no
mesh.25x25.6	62	29.56	sí
mesh.25x25.7	62	24.84	sí
mesh.25x25.8	62	43.18	sí
mesh.25x25.9	47	94.30	no
mesh.25x25.10	62	35.43	sí
mesh.25x25.11	62	34.73	sí
mesh.25x25.12	62	19.53	sí
mesh.25x25.13	62	27.30	sí
mesh.25x25.14	62	44.88	sí
mesh.25x25.15	43	94.54	no
mesh.25x25.16	46	94.33	no
mesh.25x25.17	47	94.27	no
mesh.25x25.18	62	19.35	sí
mesh.25x25.19	62	19.35	sí
mesh.25x25.20	43	99.36	no
Media	56.25	52.59	

Tabla A.11: Descripción de los resultados detallados por cada instancia, y valores promedios para el experimento con el grafo `mesh.25x25` y 62 pares terminales de conexión.

El tiempo promedio para cada instancia para el experimento total es $t_{\lambda-SP} = 2{,}28s$.

Experimento con el grafo `mesh.25x25` y 156 pares terminales

La intención de esta prueba es conexionar $K = 156$ pares terminales. El tiempo total de cómputo para cada instancia es el siguiente:

$$t_{max} = 234{,}00s$$

.

Y los resultados aparecen en la tabla A.12

El tiempo promedio para cada instancia es $t_{\lambda-SP} = 16{,}01s$.

Experimento con el grafo `mesh.25x25` y 250 pares terminales

En este último ensayo de la primera batería de pruebas, la meta es conectar $K = 250$ pares terminales. El tiempo total de cómputo para cada instancia es el siguiente:

$$t_{max} = 375{,}00s$$

Instancia	z_{EDP}	t(s)	óptimo
mesh.25x25.1	73	238.97	no
mesh.25x25.2	78	237.28	no
mesh.25x25.3	71	237.16	no
mesh.25x25.4	74	237.52	no
mesh.25x25.5	85	237.30	no
mesh.25x25.6	85	237.31	no
mesh.25x25.7	76	237.49	no
mesh.25x25.8	76	237.29	no
mesh.25x25.9	81	237.77	no
mesh.25x25.10	74	236.94	no
mesh.25x25.11	75	237.21	no
mesh.25x25.12	74	237.01	no
mesh.25x25.13	77	237.35	no
mesh.25x25.14	79	237.16	no
mesh.25x25.15	78	237.57	no
mesh.25x25.16	78	237.13	no
mesh.25x25.17	73	237.25	no
mesh.25x25.18	73	237.84	no
mesh.25x25.19	75	238.11	no
mesh.25x25.20	75	237.27	no
Media	76.50	237.45	

Tabla A.12: Descripción de los resultados detallados por cada instancia, y valores promedios para el experimento con el grafo `mesh.25x25` y 156 pares terminales de conexión.

.

Y los resultados aparecen en la tabla A.13

Instancia	z_{EDP}	t(s)	óptimo
mesh.25x25.1	87	384.58	no
mesh.25x25.2	99	383.53	no
mesh.25x25.3	98	382.15	no
mesh.25x25.4	91	381.18	no
mesh.25x25.5	87	381.99	no
mesh.25x25.6	92	382.00	no
mesh.25x25.7	95	383.48	no
mesh.25x25.8	95	381.51	no
mesh.25x25.9	92	381.01	no
mesh.25x25.10	93	383.36	no
mesh.25x25.11	89	383.07	no
mesh.25x25.12	90	381.51	no
mesh.25x25.13	94	380.49	no
mesh.25x25.14	99	381.24	no
mesh.25x25.15	93	382.18	no
mesh.25x25.16	98	380.45	no
mesh.25x25.17	96	380.31	no
mesh.25x25.18	93	380.6	no
mesh.25x25.19	102	383.56	no
mesh.25x25.20	98	380.69	no
Media	94.05	381.94	

Tabla A.13: Descripción de los resultados detallados por cada instancia, y valores promedios para el experimento con el grafo `mesh.25x25` y 250 pares terminales de conexión.

El tiempo promedio para cada instancia es $t_{\lambda-SP} = 26{,}53s$.

A.2.2. Segunda batería de pruebas

Para el segundo *benchmark* van a aparecer en las tablas los valores promedios alcanzados por cada experimento o ensayo realizado. En este caso, cada experimento se forma a partir de la ejecución de 20 veces la misma instancia que pertenece al ensayo. En cada tabla aparecerá, por tanto, estos tres valores:

- Experimento: Ensayo realizado mediante una instancia formada por un grafo y un número de pares terminales a conectar determinado. Los resultados de la prueba se genera, a partir de una ejecución repetida de la instancia 20 veces.
- z_{EDP}: El valor de la función objetivo promedio para un instancia del experimento.
- $t(s)$: El tiempo promedio de cómputo en segundos para una instancia del ensayo.
- $t_{\lambda-SP}(s)$: El tiempo de ejecución promedio del módulo $\lambda - SP$ para cada instancia.

- $t_{max}(s)$. Es el tiempo límite calculado para cada instancia del experimento. Emplea la fórmula $t_{max} = \rho \cdot V \cdot K$. Si se supera este tiempo, el algoritmo debe finalizar la iteración actual y parar. Por tanto, el método podría sobrepasar este valor.
- *óptimos*: Indica el número de veces que la instancia, sobre un valor máximo de 20, alcanza el valor óptimo en la primera fase del procedimiento; es decir, con el modelo ILP.

Experimentos con el grafo `bl-wr2-wht2.10-50.rand`

Los resultados aparecen en la tabla A.14. El número de pares terminales utilizado en cada prueba aparece en la tabla 6.3 del capítulo 6. Asímismo, el número de vértices $|V|$ del grafo es 500, tal y como aparece en la tabla 6.1 de capítulo citado en las líneas anteriores.

Experimento	z_{EDP}	$t(s)$	$t_{\lambda-SP}(s)$	$t_{max}(s)$	óptimos
1.bl.rand.10.1	12.00	62.37	6.80	60.00	0
2.bl.rand.25.1	28.74	154.19	13.87	150.00	0
3.bl.rand.40.1	34.70	244.40	28.08	240.00	0
4.bl.rand.10.2	50.00	24.85	0.00	60.00	20
5.bl.rand.25.2	30.95	152.77	15.53	150.00	0
6.bl.rand.40.2	35.65	244.33	22.41	240.00	0
Media	32.00	147.15			

Tabla A.14: Descripción de los resultados detallados por cada experimento con el grafo `bl-wr2-wht2.10-50.rand`.

Experimentos con el grafo `bl-wr2-wht2.10.50.sdeg`

Los valores alcanzados se representan en la tabla A.15. El número de pares terminales asociados a cada ensayo aparece en la tabla 6.3 del capítulo 6. Además, se indica que el número de nodos $|V|$ del grafo es 500 (ver en la tabla 6.1 del capítulo indicado previamente).

Experimento	z_{EDP}	$t(s)$	$t_{\lambda-SP}(s)$	$t_{max}(s)$	óptimos
7.bl.sdeg.10.1	47.00	15.36	0.00	60.00	20
8.bl.sdeg.25.1	103.20	76.80	0.88	150.00	19
9.bl.sdeg.40.1	33.45	244.55	23.82	240.00	0
10.bl.sdeg.10.2	50.00	13.90	0.00	60.00	20
11.bl.sdeg.25.2	29.60	152.93	16.01	150.00	0
12.bl.sdeg.40.2	34.75	244.59	24.24	240.00	0
Media	49.67	124.69			

Tabla A.15: Descripción de los resultados detallados por cada experimento con el grafo `bl-wr2-wht2.10-50.sdeg`.

Experimentos con el grafo `mesh15x15`

La salida de las pruebas se exponen en la tabla A.16. En estos experimentos, el número de pares terminales involucrados en cada uno de ellos se indican en la tabla 6.4 del capítulo 6 del

documento. En la tabla 6.1 del mismo capítulo se indica que el número de vértices $|V|$ del grafo involucrado en los experimentos es 225.

Experimento	z_{EDP}	$t(s)$	$t_{\lambda-SP}(s)$	$t_{max}(s)$	óptimos
13.mesh.15.10.1	11.95	12.20	2.09	12.00	0
14.mesh.15.25.1	18.95	30.39	3.37	30.00	0
15.mesh.15.40.1	23.95	49.31	4.88	48.60	0
16.mesh.15.10.2	12.20	12.18	1.59	12.00	0
17.mesh.15.25.2	22.10	30.34	4.41	30.00	0
18.mesh.15.40.2	25.85	49.53	5.21	48.60	0
Media	19.17	30.67			

Tabla A.16: Descripción de los resultados detallados por cada experimento con el grafo `mesh15x15`.

Experimentos con el grafo `mesh25x25`

La tabla A.17 ofrece los resultados relativos a estos ensayos. El número de pares terminales empleado en cada prueba aparecen en la tabla 6.4 del capítulo 6. El número de vértices $|V|$ del grafo es 625. Este valor se extrae de la tabla 6.1 del capítulo mencionado anteriormente.

Experimento	z_{EDP}	$t(s)$	$t_{\lambda-SP}(s)$	$t_{max}(s)$	óptimos
19.mesh.25.10.1	62.00	42.25	0.00	93.00	20
20.mesh.25.25.1	29.30	238.36	29.17	234.00	0
21.mesh.25.40.1	37.90	382.42	40.94	375.00	0
22.mesh.25.10.2	62.00	52.75	0.00	93.00	20
23.mesh.25.25.2	30.25	238.64	31.11	234.00	0
24.mesh.25.40.2	35.50	382.72	51.00	375.00	0
Media	42.83	222.86			

Tabla A.17: Descripción de los resultados detallados por cada experimento con el grafo `mesh25x25`.

Experimentos con el grafo `steinb10`

Los resultados aparecen en la tabla A.18. El número de pares terminales manejados en cada ensayo se describe en la tabla 6.6 del capítulo 6. El número de nodos $|V|$ del grafo es 75. Esta información se indica en la tabla 6.1.

Experimentos con el grafo `steinb16`

Los valores de los ensayos se describen en la tabla A.19. El número de pares terminales empleado en cada prueba se indican en la tabla 6.6 del capítulo 6. El número de vértices $|V|$ del grafo es 100, tomándose tal valor de la tabla 6.1 del capítulo de resultados.

Experimento	z_{EDP}	$t(s)$	$t_{\lambda-SP}(s)$	$t_{max}(s)$	óptimos
28.steinb10.10	7.00	0.38	0.07	1.26	18
29.steinb10.25	18.00	4.12	0.84	3.24	0
30.steinb10.40	28.00	5.52	1.00	5.40	0
Media	17.67	3.34			

Tabla A.18: Descripción de los resultados detallados por cada experimento con el grafo `steinb10`.

Experimento	z_{EDP}	$t(s)$	$t_{\lambda-SP}(s)$	$t_{max}(s)$	óptimos
31.steinb16.10	10.00	1.09	0.17	2.40	17
32.steinb16.25	25.00	6.43	1.06	6.00	0
33.steinb16.40	36.00	85.75	1.79	9.60	0
Media	23.67	31.09			

Tabla A.19: Descripción de los resultados detallados por cada experimento con el grafo `steinb16`.

Experimentos con el grafo `steinc6`

La salida aparece en la tabla A.20; donde el número de pares terminales utilizado en cada prueba se muestra en la tabla 6.7 del capítulo 6. Por otra parte, la cantidad de vértices $|V|$ del grafo es 500, tal y como se describe en la tabla 6.1.

Experimento	z_{EDP}	$t(s)$	$t_{\lambda-SP}(s)$	$t_{max}(s)$	óptimos
34.steinc6.10	50.00	4.62	0.00	60.00	20
35.steinc6.25	125.00	14.16	0.00	150.00	20
36.steinc6.40	200.00	23.22	0.00	240.00	20
Media	125.00	14.00			

Tabla A.20: Descripción de los resultados detallados por cada experimento con el grafo `steinc6`.

Experimentos con el grafo `steinc11`

Los resultados de cada ensayo se muestra en la tabla A.21. El número de pares terminales asociado a cada prueba se toma de la tabla 6.7 incluido en el capítulo 6. Y el valor $|V|$ del grafo es también 500, adquiriendo este número de la tabla 6.1.

Experimentos con el grafo `steinc16`

Los valores de las pruebas se representan en la tabla A.22. El número de pares terminales de cada ensayo aparece en la tabla 6.7 del capítulo 6; y el de vértices $|V|$ del grafo es también 500 (ver la tabla 6.1).

Hay que destacar, en este caso, la gran cantidad de tiempo de cómputo necesario para obtener los $\lambda-SP$ caminos más cortos en los experimentos `41.steinc16.25` y `42.steinc16.40` (840.8s y 1305.5s respectivamente). El grafo contiene 500 nodos y 12500 enlaces. Se considera

Experimento	z_{EDP}	$t(s)$	$t_{\lambda-SP}(s)$	$t_{max}(s)$	óptimos
37.steinc11.10	50.00	15.84	0.00	60.00	20
38.steinc11.25	125.00	30.89	0.00	150.00	20
39.steinc11.40	200.00	54.23	0.00	240.00	20
Media	125.00	33.65			

Tabla A.21: Descripción de los resultados detallados por cada experimento con el grafo `steinc11`.

Experimento	z_{EDP}	$t(s)$	$t_{\lambda-SP}(s)$	$t_{max}(s)$	óptimos
40.steinc16.10	50.00	96.00	0.00	60.00	20
41.steinc16.25	125.00	1064.35	840.80	150.00	0
42.steinc16.40	200.00	1714.83	1305.50	240.00	0
Media	125.00	958.39			

Tabla A.22: Descripción de los resultados detallados por cada experimento con el grafo `steinc16`.

que debido a que este último valor es tan alto, el tiempo necesario para generar los caminos se incremente considerablemente.

Experimentos con el grafo `planar.50`

Los resultados de los ensayos se indican en la tabla A.23. El número de pares terminales utilizado en cada prueba se toman de la tabla 6.5 del capítulo 6. Asímismo, el número de vértices $|V|$ del grafo es 50 (tabla 6.1).

Experimento	z_{EDP}	$t(s)$	$t_{\lambda-SP}(s)$	$t_{max}(s)$	óptimos
43.planar.50.10	5.00	0.21	0.06	0.60	18
44.planar.50.25	12.00	1.05	0.18	1.44	17
45.planar.50.40	20.00	4.32	0.84	2.40	0
Media	12.33	1.86			

Tabla A.23: Descripción de los resultados detallados por cada experimento con el grafo `planar.50`.

Experimentos con el grafo `planar.100`

Los resultados aparecen en la tabla A.24. En este caso, se emplea un grafo plano con 100 nodos (tabla 6.1). Los pares terminales que se deben conectar en casa ensayo se toman de la tabla 6.5.

Experimentos con el grafo `planar.200`

Los resultados se muestran en la tabla A.25. También se emplea un grafo plano, pero conteniendo 200 nodos (tabla 6.1). El número de pares terminales a conexionar se indica en la tabla 6.5.

Experimento	z_{EDP}	$t(s)$	$t_{\lambda-SP}(s)$	$t_{max}(s)$	óptimos
46.planar.100.10	10.00	1.15	0.14	2.40	18
47.planar.100.25	24.90	2.55	0.20	6.00	18
48.planar.100.40	33.10	9.89	2.26	9.60	0
Media	22.67	4.53			

Tabla A.24: Descripción de los resultados detallados por cada experimento con el grafo `planar.100`.

Experimento	z_{EDP}	$t(s)$	$t_{\lambda-SP}(s)$	$t_{max}(s)$	óptimos
49.planar.200.10	20.00	2.55	0.00	9.60	20
50.planar.200.25	50.00	5.83	0.00	24.00	20
51.planar.200.40	80.00	12.46	0.00	38.40	20
Media	50.00	6.95			

Tabla A.25: Descripción de los resultados detallados por cada experimento con el grafo `planar.200`.

Experimentos con el grafo `planar.500`

Los valores de salida de último experimento de la segunda batería de pruebas aparecen en la tabla A.26. Se utiliza un grafo plano con 500 nodos (tabla 6.1). El número de pares terminales a conectar se describe en la tabla 6.5.

Experimento	z_{EDP}	$t(s)$	$t_{\lambda-SP}(s)$	$t_{max}(s)$	óptimos
52.planar.500.10	50.00	5.62	0.00	60.00	20
53.planar.500.25	125.00	17.12	0.00	140.00	20
54.planar.500.40	200.00	33.07	0.00	240.00	20
Media	50.00	6.95			

Tabla A.26: Descripción de los resultados detallados por cada experimento con el grafo `planar.500`.

Apéndice B

Contribuciones de la investigación

B.1. Introducción

A continuación se van a indicar los contribuciones científicas que se han generado a partir de las investigaciones realizadas y descritas en esta memoria.

B.2. Aportaciones

Se incluyen los siguientes articulos aceptados en congresos y revista *Journal Citation Reports*:

- Bernardo Martín, Ángel Sánchez, César Beltrán-Royo and Abraham Duarte. Solving the Edge-Disjoint Paths Problem using a two-stage method. *International Transactions in Operational Research*, DOI: 10.1111/itor.12544. Received 11 November 2016; received in revised form 16 March 2018; accepted 24 March 2018; first published 30 April 2018.
- Bernardo Martín, Ángel Sánchez, César Beltrán-Royo and Abraham Duarte. A Matheuristic Approach for Solving the Edge-Disjoint Paths Problem. *Matheuristics 2016 - Proceedings of the Sixth International Workshop on Model-based Metaheuristics*, pages 25-34, 2016.
- B. Martín, A. Sánchez y A. Duarte. Resolución del problema EDP para Grafos No Dirigidos utilizando algoritmos MSGA y GRASP. *IV Congreso Español de Informática, IX Congreso Español de Metaheursticas, Algoritmos Evolutivos y Bioinspirados*, páginas 557-566, 2013.
- B. Martín, A. Sánchez y A. Duarte. Resolución del problema de los caminos disjuntos en grafos dirigidos usando técnicas metaheurísticas. *II Congreso Español de Informática, I Jornada sobre Algoritmos Evolutivos y Metaheurísticas*, 2007.

Bibliografía

[1] Scott Aaronson. https://www.scottaaronson.com/blog/?p=122.

[2] R. K. Ahuja, O. Ergun, J. B. Orlin, and A. P. Punnen. A survey of very large-scale neighborhood search techniques. *Discrete Applied Mathematics*, 123 (1-3):75–102, 2002.

[3] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, 1993.

[4] Mohan Akella, Sharad Gupta, and Avijit Sarkar. Branch and price: Column generation for solving huge integer programs. Technical report, Department of Industrial Engineering. University at Buffalo (SUNY).

[5] E. Alba, F. Almeida, M. Blesa, C. Cotta, M. Dfaz, I. Dorta, J. Abarr, J. González, C. Len, L. Moreno, J. Petit, J. Roda, A. Rojas, and F. Xhafa. Mallba: A library of skeletons for combinatorial optimisation. In *Proceedings of the Euro-Par*, volume 2004 of LNCS, pages 927–932, 2002.

[6] Enrique Alba, Gabriel Luque, and Sergio Nesmachnow. Parallel metaheuristics: recent advances and new trends. *International Transactions in Operational Research*, 20(1):1–48, 2013.

[7] Unai Aldasoro, Laureano F. Escudero, María Merino, and Gloria Pérez. A parallel branch-and-fix coordination based matheuristic algorithm for solving large sized multistage stochastic mixed 01 problems. *European Journal of Operational Research*, 258:590–606, 2017.

[8] Aldeida Aleti, Barbora Buhnova, Lars Grunske, Anne Koziolek, and Indika Meedeniya. Software architecture optimization methods: A systematic literature review. *IEEE Transactions on Software Engineering*, 39 (5), 2013.

[9] Noha Alsulami and Mohamed Khamis. Solutions of routing and wavelength assignment in wavelength division multiplexing optical network: A survey. *Journal of Emerging Trends in Computing and Information Sciences*, 6 (5), 2015.

[10] F. Altarelli, A. Braunstein, L. DallAsta, C. De Bacco, and S. Franz. The edge-disjoint path problem on random graphs by message-passing. *PLoS One*, 10(12), 2015.

[11] Fabrizio Altarelli, Alfredo Braunstein, John Realpe-Gomez, and Riccardo Zecchina. Statistical mechanics of budget-constrained auctions. *Journal of Statistical Mechanics: Theory and Experiment*, 7, 2009.

[12] M. Andrews, J. Chuzhoy, K. Sanjeev, and L. Zhang. Hardness of the undirected edge-disjoint paths problem with congestion. In *Proc. 46th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2005)*, pages 226–241, 2005.

[13] I. Androulakis, C. Maranas, and C. Floudas. A global optimization method for general constrained nonconvex problems. *Journal of Global Optimization*, 7:337–363, 1995.

[14] D. Applegate, R. Bixby, V. Chvátal, and W. Cook. Finding tours in the tsp. Technical report, Report 99885, Forschungsinstitut fur Diskrete Mathematik, University of Bonn, Germany, 1999.

[15] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proff verification and the hardness of approximation problems. *Journal of the ACM*, 1998.

[16] P. Augerat, J.M. Belenguer, E. Benavent, A. Corberan, and D. Naddef. Separating capacity constraints in the cvrp using tabu search. *European Journal of Operational Research*, 106:546–557, 1998.

[17] Y. Aumann and Y. Rabani. Approximation algorithms for edge-disjoint paths and unsplittable flow. In *Proceedings of the sixth annual ACM-SIAM symposium on Discrete algorithms, SODA'95*, pages 567–576, 2006.

[18] G. Ausiello, M. Protasi, A. Marchetti-Spaccamela, G. Gambosi, P. Crescenzi, and V. Kann. *Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties.* Springer-Verlag New York, Inc., 1999.

[19] B. Awerbuch, Y. Azar, and S. Plotkin. Throughtput-competitive online routing. *34th Annual IEEE Symposium on Foundations of Computer Science*, 1993.

[20] B. Awerbuch, R. Gawlick, F.T. Leighton, and Y. Rabani. On-line admission control and circuit routing for high performance computing and communication. In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science (FOCS94)*, pages 412–423, 1994.

[21] C. De Bacco, S. Franz, D. Saad, and C. H. Yeung. Shortest node-disjoint paths on random graphs. *Journal of Statistical Mechanics: Theory and Experiment*, 7, 2014.

[22] S. Baluja. Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning. *The International Conference on Machine Learning. Eds. Morgan-Kaufmann Publishers*, 1995.

[23] S. Baluja and R. Caruana. Removing the genetics from the standard genetic algorithm. *The International Conference on Machine Learning. Eds. Morgan-Kaufmann Publishers*, 1995.

[24] J. Bang-Jensen and G. Gutin. *Digraphs: Theory, Algorithms and Applications*. Springer, 2001.

[25] A. Barabasi and R. Albert. Emergence of scaling in random networks. *Science*, 286, 1999.

[26] L. Barcos, V. M. Rodríguez, M. J. Álvarez, and F. Robusté. Algoritmo basado en la optimización mediante colonias de hormigas para la resolución del problema del transporte de carga desde varios orígenes a varios destinos. *V Congreso de Ingeniera del Transporte. CIT*, 2002.

[27] M. Basseur, J. Lemesre, and E.G. Talbi C. Dhaenens. Cooperation between branch and bound and evolutionary approaches to solve a biobjective flow shop problem. *Workshop on Evolutionary Algorithms (WEA04), Springer-Verlag*, 3059:72–86, 2004.

[28] A. Baveja and A. Srinivasan. Approximation algorithms for disjoint paths and related routing and packing problems. *Mathematics of Operations Research*, 25(2), 2000.

[29] R. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, 1957.

[30] A. Bendjoudi, S. Guerdah, M. Mansoura, N. Melab, and E.G. Talbi. Metaheuristics for scheduling in distributed computing environments, chapter p2p b&b and ga for the flow-shop scheduling problem. *Studies in Computational Intelligence, Springer*, 2008.

[31] R. Bent and P. Van Hentenryck. A two-stage hybrid local search for the vehicle routing problem with time windows. Technical report, Tech Report CS-01-06, Brown University, USA, 2001.

[32] D. P. Bertsekas. *Network Optimization: Continuous and Discrete Models*. Athena Scientific, Belmont, MA, 1998.

[33] M. J. Blesa and C. Blum. Ant colony optimization for the maximum edge-disjoint paths problem. *1st European Workshop on Evolutionary Computation in Communication Networks, and Connected Systems*, 2004.

[34] M. J. Blesa and C. Blum. Finding edge-disjoint paths in networks by means of artificial ant colonies. *Journal of Mathematical Modeling and Algorithms*, 6 (3), 2007.

[35] C. Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys (CSUR)*, 35 (3), 2003.

[36] Christian Blum, Jakob Puchinger, Günther Raidl, and Andrea Roli. Hybrid metaheuristics in combinatorial optimization: A survey. *Applied Soft Computing, Elsevier*, 11 (6):4135–4151, 2011.

[37] U. Brandes and D. Wagner. A linear time algorithm for the arc disjoint menger problem in planar directed graphs. *Algorithmica*, 28 (1), 2000.

[38] G. Brassard and P. Bratley. *Fundamentals of Algorithmics*. Prentice Hall, 1996.

[39] Alfredo Braunstein and Riccardo Zecchina. Learning by message passing in networks of discrete synapses. *Physical review letters*, 96 (3), 2006.

[40] A. Broder, A. Frieze, and E. Upfal. Existence and construction of edge-disjoint paths on expander paths. *SIAM Journal on Computing*, 23, 1994.

[41] K. Büdenbender, T. Grünert, and H. J. Sebastian. A hybrid tabu search/branch-andbound algorithm for the direct flight network design problem. *Transportation Science*, 34 (4):364–380, 2000.

[42] E. K. Burke, P. Cowling, and R. Keuthen. Embedded local search and variable neighbourhood search heuristics applied to the travelling salesman problem. Technical report, University of Nottingham. U.K., 2000.

[43] E. K. Burke, P. I. Cowling, and R. Keuthen. Effective local and guided variable neighbourhood search methods for the asymmetric travelling salesman problem. *EvoWorkshop 2001, LNCS*, 2037:203–212, 2001.

[44] Matthew Caesar, Tyson Condie, Jayanthkumar Kannan, Karthik Lakshminarayanan, and Ion Stoica. ROFL: Routing on flat labels. In *ACM SIGCOMM*, 2006.

[45] M. Cárdenas. Eda - pbil. Technical report, Centro de Investigaciones Energticas Medioambientales y Tecnolgicas. CIEMAT, Madrid, 2012.

[46] J. Carlier. The one-machine sequencing problem. *European Journal of Operational Research*, 1982.

[47] P. Carmi, T. Erlebach, and Y. Okamoto. Greedy edge-disjoint paths in complete graphs. *ETH Zurich, the German Science Foundation (DFG)*, 6 (3), 2001.

[48] Y. Caseau and F. Laburthe. Effective forget-and-extend heuristics for scheduling problems. *CP-AI-OR02 Fourth International Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problem*, 1999.

[49] A. Chabrier, E. Danna, and C. Le Pape. Coopération entre génération de colonnes sans cycle et recherche locale appliquée au routage de véhicules. *JNPC*, 2002.

[50] W. T. Chan, F. Y. L. Chin, and H. F. Ting. Escaping a grid by edge-disjoint paths. *Algorithmica*, 36(4), 2012.

[51] C. Chekuri and S. Khanna. Edge disjoint paths revisited. In *Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'03)*, pages 628–637, 2003.

[52] R. Chelouah and P. Siarry. A hybrid method combining continuous tabu search and nelder-mead simplex algorithms for the global optimization of multiminima functions. *European Journal of Operational Research*, 161 (3):636–654, 2005.

[53] S. Chen, S. Talukdar, and N. Sadeh. Job-shop-scheduling by a team of asynchronous agents. In *IJCAI-93 Workshop on Knowledge-Based Production, Scheduling and Control, Chambery, France*, 1993.

[54] J. F. Chicano. *Metaheurísticas e Ingeniería del Software*. PhD thesis, Dpto. Lenguajes y Ciencias de la Computación. Universidad de Málaga, 2007.

[55] J. W. Chinneck. Practical optimization: A gentle introduction. Technical report, Systems and Computer Engineering. Carleton University. Ottawa, Ontario K1S 5B6. Canada. Available: http://www.sce.carleton.ca/faculty/chinneck/po.html., 2015.

[56] I. Chlamtac, A. Ganz, and G. Karmi. "lightpath communications: an approach to high bandwidth optical wan's. *IEEE Transactions on Communications*, 40 (7):1171–1182, 1992.

[57] E. K. P. Chong and S. H. Zak. *An introduction to Optimization*. John Wiley and Sons, 2008.

[58] P. C. Chu and J. E. Beasley. A genetic algorithm for the multidimensional knapsack problem. *Journal of Heuristics*, 4:63–86, 1998.

[59] J. Chuzhoy. Approximation algorithms for graph routing problems. Technical report, Toyota Technological Institute, Chicago.

[60] D. Clements, J. Crawford, D. Joslin, G. Nemhauser, M. Puttlitz, and M. Savelsbergh. Heuristic optimization: A hybrid ai/or approach. In *Proceedings of the Workshop on Industrial Constraint-Directed Scheduling*, 1997.

[61] R. K. Congram. *Polynomially Searchable Exponential Neighbourhoods for Sequencing Problems in Combinatorial Optimisation*. PhD thesis, University of Southampton, Faculty of Mathematical Studies, U.K., 2000.

[62] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceeding STOC '71 Proceedings of the third annual ACM symposium on Theory of computing*, 1971.

[63] O. Cordón. Sistemas complejos, algoritmos evolutivos y bioinspirados. parte i: Algoritmos de optimizacin basados en colonia de hormigas. Technical report.

[64] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms (2nd ed.)*. MIT Press and McGrawHill, 1990 (2001).

[65] L. E. Corredera. Simulación de comportamiento de una colonia de hormigas. Technical report, Universidad Pontificia de Salamanca en Madrid.

[66] C. Cotta, J.F. Aldana, A.J. Nebro, and J.M. Troya. Hybridizing genetic algorithms with branch and bound techniques for the resolution of the tsp. In *D.W. Pearson, N.C. Steele, R.F. Albrecht (Eds.), Artificial Neural Nets and Genetic Algorithms*, volume 2, pages 277–280, 1995.

[67] C. Cotta and J. M. Troya. Embedding branch and bound within evolutionary algorithms. *Applied Intelligence*, 18:137–153, 2003.

[68] Pierluigi Crescenzi and Viggo Kann. A compendium of np optimization problems. Technical report, KTH Royal Institute of Technology. Sweden. Available: http://www.nada.kth.se/theory/compendium/, 1998.

[69] Harlan Crowder, Ellis L. Johnson, and Manfred Padberg. Solving large-scale zero-one linear programming problem. *Operations Research*, 31 (5):803–834, 1983.

[70] Patrick W. Daly. *Naural Sciences Citations and References*, February 2007.

[71] George Dantzig and Mukund Thapa. *Linear Programming 2: Theory and Extensions.* Springer-Verlag, 2003.

[72] J. Denzinger and T. Offermann. On cooperation between evolutionary algorithms and other search paradigms. In *Proceedings of the 1999 Congress on Evolutionary Computation (CEC). IEEE Press*, pages 2317–2324, 1999.

[73] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematlk*, 1:269–271, 1959.

[74] M. Dorigo, D. Corne, and F. Glover. *New Ideas in Optimization.* McGraw Hill, 1999.

[75] Marco Dorigo. *Optimization, Learning and Natural Algorithms (in Italian).* PhD thesis, Dipartimento di Elettronica, Politecnico di Milano, 1992.

[76] S. Droste, T. Jansen, and I. Wegener. Optimization with randomized search heuristics: the (a)nfl theorem, realistic scenarios, and difficult functions. *Theoretical Computer Science*, 287 (1):131–144, 2002.

[77] A. Duarte, J. J. Pantrigo, and M. Gallego. *Metaheurísticas.* Dykinson, 2007.

[78] Abraham Duarte. *Algoritmos Sociales Jerárquicos: Una metaheurística basada en la hibridación entre métodos constructivos y evolutivos.* PhD thesis, Universidad Rey Juan Carlos, 2004.

[79] I. Dumitrescu and T. Stützle. Combinations of local search and exact algorithms. *EvoWorkshops 2003*, pages 211–223, 2003.

[80] P.Q. Dung, Y. Deville, and P. Van Hentenryck. A constraint-based local search for constraint optimization on trees and paths. *Constraints*, 17 (4):357–408, 2012.

[81] R. C. Eberhart and J. Kennedy. A new optimizer using particle swarm theory. In *Proceedings of the Sixth International Symposium on Micromachine and Human Science, Nagoya, Japan*, 1995.

[82] T. English. No more lunch: Analysis of sequential search. In *Proceedings of the 2004 IEEE Congress on Evolutionary Computation*, 2004.

[83] T. Erlebach. Approximation algorithms and complexity results for path problems in trees of rings. *26th International Symposium of Mathematical Foundations of Computer Science. Springer-Verlag*, LNCS 2136:351–362, 2001.

[84] T. Erlebach and K. Jansen. The maximum edge-disjoint paths problem in bidirected trees. *SIAM Journal on Discrete Mathematics*, 14 (3):326–355, 2001.

[85] T. Erlebach and D. Vukadinovic. New results for path problems in generalized stars, complete graphs, and brick wall graphs. In *Proceedings of the 13th International Symposium on Fundamentals of Computation Theory (FCT 2001)*, volume LNCS 2138, pages 483–494, 2001.

[86] Thomas Erlebach. Approximation: Theory and algorithms. edge-disjoint paths and unsplittable flow. Technical report, ETH Zrich. Available: *https* : *//www.ti.inf.ethz.ch/ew/courses/ApproxAlgs*0304*/problem_sets/edpsurvey.pdf*, 1993.

[87] Thomas Erlebach. Approximation algorithms for edge-disjoint paths and unsplittable flow. In *E. Bampis et al. (Eds.): Efficient Approximation and Online Algorithms. Springer-Verlag*, volume LNCS 3484, pages 97–134, 2006.

[88] Real Academia Española. *Diccionario de la lengua española. Edición del Tricentenario. http://dle.rae.es/?w=diccionario.* 2014.

[89] S. Even, A. Itai, and A. Shamir. On the complexity of timetable and multicommodity flow problems. *SIAM Journal on Computing*, 5 (4), 1976.

[90] G. R. Filho and L. A. N. Lorena. Constructive genetic algorithm and column generation: an application to graph coloring. In *Proceedings of APORS 2000 - The Fifth Conference of the Association of Asian-Pacific Operations Research Societies within IFORS*, 2000.

[91] P. R. Fillottrani. Algoritmos y complejidad. complejidad computacional. Technical report, Dpto. Ciencias e Ingeniería de la Computación. Universidad Nacional del Sur, 2011.

[92] C. A. Floudas and P. M. Pardalos. *Encyclopedia of Optimization.* Springer, 2009.

[93] S. Fortune, J. Hopcroft, and J. Wyllie. The directed subgraph homeomorphism problem. *Theoretical Computer Science. North-Holland Publishing Company,*, 10 (2):111–121, 1980.

[94] A. Frank. Packing paths, circuits, and cuts. a survey. In *B. Korte, L. Lovász, H. J. Prömel, and A. Schrijver, editors, Paths, Flows, and VLSI-Layout. Springer-Verlag*, pages 47–100, 1990.

[95] A. P. French, A. C. Robinson, and J. M.Wilson. Using a hybrid geneticalgorithm/ branch and bound approach to solve feasibility and optimization integer programming problems. *Journal of Heuristics*, 7:551–564, 2001.

[96] Micael Gallego. *Resolución exacta y aproximada del Problema de la Diversidad Mxima.* PhD thesis, Universidad Rey Juan Carlos, 2008.

[97] Á. García. Técnicas metaheurísticas. Technical report, ETSI Industriales. Universidad Politécnica de Madrid.

[98] Eduardo García. *El problema de la minimización de la anchura de corte en ordenaciones lineales: resolución exacta y heurística.* PhD thesis, Universidad Rey Juan Carlos, 2011.

[99] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman, 1979.

[100] N. Garg, V. Vazirani, and M. Yannakakis. Primal-dual approximation algorithms for integral flow and multicut in trees. *Algorithmica*, 18 (1):3–20, 1997.

[101] P. C. Gilmore. Optimal and suboptimal algorithms for the quadratic assignment problem. *Journal of the SIAM*, 10:305–313, 1962.

[102] F. Glover. Heuristics for integer programming using surrogate constraints. *Dec. Sci.*, 8:156–166, 1977.

[103] F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research*, 13(5), 1986.

[104] F. Glover. Tabu search-part i. *ORSA Journal on Computing*, 1:190–206, 1989.

[105] F. Glover. Tabu search-part ii. *ORSA Journal on Computing*, 2:4–32, 1990.

[106] F. Glover and G. Kochenberger. *Handbook of Metaheuristics, volume 57*. International Series in Operations Research & Management Science. Kluwer Academic Publishers, 2003.

[107] F. Glover and M. Laguna. *Tabu search*. Kluwer Academic Publishers, Boston, 1997.

[108] M. Gómez. *Sistema de generacin eléctrica con pila de combustible de óxido slido alimentado con resíduos forestales y su optimización mediante algoritmos basados en nubes de partículas*. PhD thesis, Departamento de Ingeniería Eléctrica, Electrónico y de Control. UNED, 2008.

[109] C. González, J. A. Lozano, and P. Larrañaga. Analyzing the population based incremental learning algorithm by means of discrete dynamical systems. *Complex Systems Publications, Inc.*, 12:465–479, 2000.

[110] R. Guerequeta and A. Vallecillo. Técnicas de diseño de algoritmos. Technical report, Servicio de Publicaciones de la Universidad de Málaga, 1998.

[111] U. I. Gupta, D. T. Lee, and J. Y. T. Leung. Efficient algorithms for interval graphs and circular-arc graphs. *Networks*, 12:459–467, 1982.

[112] V. Guruswami, S. Khanna, R. Rajaraman, B. Shepherd, and M. Yannakakis. Nearoptimal hardness results and approximation algorithms for edge-disjoint paths and related problems. In *Proceedings of the 31st Annual ACM Symposium on Theory of Computing (STOC99)*, pages 19–28, 1999.

[113] V. Guruswami, S. Khanna, R. Rajaraman, B. Shepherd, and M. Yannakakis. Near-optimal hardness results and approximation algorithms for edge-disjoint paths and related problems. *Journal of Computer and System Sciences*, 2003.

[114] P. Hansen and N. Mladenovic. An introduction to variable neighborhood search. In *Metaheuristics: Advances and trends in local search paradigms for optimization. Kluwer Academic Publishers*, 1999.

[115] P. Hansen and N. Mladenovic. Variable neighborhood search: Principles and applications. *European Journal of Operational Research*, 130:449–467, 2001.

[116] Ali Hassan, Chris Phillips, and Jonathan Pitts. Dynamic routing and wavelength assignment using hybrid particle swarm optimization for wdm networks. *EPSRC PostGraduate Network Symposium (PGNet*, 2007.

[117] K. Helsgaun. An effective implementation of the lin-kernighan traveling salesman heuristic. *European Journal of Operational Research*, 126:106–130, 2000.

[118] J. H. Holland. Adaptation in natural and artificial systems. Technical report, University of Michigan Press, 1975.

[119] C. C. Hsu, H. J. Cho, and S. C. Fang. Routing and wavelength assignment in optical networks from maximum edge-disjoint paths. *Genetic and Evolutionary Computing. Advances in Intelligent Systems and Computing. Springer*, 238, 2012.

[120] Chia-Chun Hsu and Hsun-Jung Cho. A genetic algorithm for the maximum edge-disjoint paths problem. *Neurocomputing*, 148:17–22, 2015.

[121] IBM. *IBM ILOG CPLEX Optimization Studio CPLEX Users Manual. Version 12 Release 7.* IBM Corp., 2016.

[122] C. Igel and M. Toussaint. A no-free-lunch theorem for non-uniform distributions of target functions. *Journal of Mathematical Modelling and Algorithms*, 3:313–322, 2004.

[123] C.A.R. Jahuira. Hybrid genetic algorithm with exact techniques applied to tsp. In *Second International Workshop on Intelligent Systems Design and Application, Dynamic Publishers*, pages 119–124, 2002.

[124] C.A.R. Jahuira and E. Cuadros-Vargas. Solving the tsp by mixing gas with minimal spanning tree. *First International Conference of the Peruvian Computer Society, Lima Perú, Peruvian Computer Society*, 2003.

[125] David S. Johnson. A brief history of np-completeness, 19542012. *Documenta Mathematica. Extra Volume ISMP*, 2012.

[126] D. E. Joslin and D. P. Clements. Squeaky wheel optimization. *Journal of Artificial Intelligence Research*, 10:353–373, 1999.

[127] L. Jourdan, M. Basseur, and E. G. Talbi. Hybridizing exact methods and metaheuristics: A taxonomy. *European Journal of Operational Research. Elsevier*, 199:620–629, 2009.

[128] N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4:373–395, 1984.

[129] R. Karp. On the complexity of combinational problems. *Networks*, 5, 1975.

[130] Richard M. Karp. Reducibility among combinatorial problems. In *R. E. Miller and J.W. Thatcher, editors, Complexity of Computer Computations. Plenum Press. New York*, 1972.

[131] R.M. Karp, F.T. Leighton, R. Rivest, C. Thompson, U. Vazirani, and V. Vazirani. Global wire routing in two-dimensional arrays. *Algorithmica*, 2, 1987.

[132] J. Kennedy and R. C. Eberhart. Particle swarm optimization. In *Proceedings of IEEE International Conference on Neural Networks, Piscataway, NJ*, 1995.

[133] S. Khanna. New algorithms for disjoint paths problems. Technical report, University of Pennsylvania.

[134] G. Klau, I. Ljubic, A. Moser, P. Mutzel, P. Neuner, U. Pferschy, G. Raidl, and R. Weiskircher. Combining a memetic algorithm with integer programming to solve the prize-collecting steiner tree problem. *Genetic and Evolutionary Computation GECCO 2004*, 3102:1304–1315, 2004.

[135] J. Kleinberg. *Approximation algorithms for disjoint path problems.* PhD thesis, MIT, 1996.

[136] J. Kleinberg and E. Tardos. Approximations for the disjoint paths problem in high-diameter planar networks. In *Proceedings of the 27th Annual ACM Symposium on Theory of Computing (STOC95)*, pages 26–35, 1995.

[137] J. Kleinberg and E. Tardos. Disjoint paths in densely embedded graphs. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science (FOCS95)*, pages 52–61, 1995.

[138] J.L. Klepeis, M.J. Pieja, and C.A. Floudas. Hybrid global optimization algorithms for protein structure prediction: Alternating hybrids. *Biophysical Journal*, 4 (84):869–882, 2003.

[139] D. E. Knuth. Postscript about np-hard problems. *SIGACT*, Newsletter 5(1):12–18, 1974.

[140] Donald E. Knuth. *The TEXbook.* Addison-Wesley, 1984.

[141] S. Kolliopoulos and C. Stein. Improved approximation algorithms for unsplittable flow problems. *38th Annual IEEE Symposium on Foundations of Computer Science*, 1997.

[142] S. Kolliopoulos and C. Stein. Approximating disjoint-path problems using packing integer programs. *Mathematical Programming*, 99 (1), 2004.

[143] S. G. Kolliopoulos. Edge-disjoint paths and unsplittable flow. *Handbook of Approximation Algorithms and Metaheuristics, Chapman and Hall/CRC*, 2007.

[144] S. G. Kolliopoulos and C. Stein. Approximating disjoint-path problems using greedy algorithms and packing integer programs. In *Proceedings of the 6th Integer Programming and Combinatorial Optimization Conference (IPCO VI)*, volume LNCS 1412, pages 153–168, 1998.

[145] P. Kolman and C. Scheideler. Improved bounds for the unsplittable flow problem. *13th annual ACM-SIAM symposium on Discrete algorithms (SODA'02) Society for Industrial and Applied Mathematics*, 2002.

[146] K. Kostikas and C. Fragakis. Genetic programming applied to mixed integer programming. In *M. Keijzer et al., editors, Genetic Programming - EuroGP 2004, Springer*, volume 3003 of LNCS, pages 113–124, 2004.

[147] M. Kramer and J. van Leeuwen. The complexity of wire routing and finding the minimum area layouts for arbitrary vlsi circuits. In *F. P. Preparata, editor, Advances in Computing Research; VLSI Theory. JAI Press Inc., Greenwich, CT-London*, volume 2, pages 129–146, 1984.

[148] Leslie Lamport. *LaTeX: A Document Preparation System.* Addison-Wesley, 1986.

[149] J. Laredo, P. Castillo, A. Mora, J. Merelo, and C. Fernandes. Resilience to churn of a peer-to-peer evolutionary algorithm. *Int. J. High Perform. Syst. Archit.*, 1:260–268, 1963.

[150] J. Laredo, A. Eiben, M. Steen, P. Castillo, A. Mora, and J. Merelo. P2p evolutionary algorithms: A suitable approach for tackling large instances in hard optimization problems. In *Proceedings of the 14th international Euro-Par conference on Parallel Processing, Euro-Par 08. Berlin, Heidelberg, 2008. Springer-Verlag*, pages 622–631, 2008.

[151] E. L. Lawler. The quadratic assignment problem. *Management Science*, 9:586–599, 1963.

[152] J. Lee, H. Scheraga, and S. Rackovsky. New optimization method for conformational energy calculations on polypeptides: Conformational space annealing. *Journal of Computational Chemistry*, 18:1222–1232, 1997.

[153] T. Lengauer. *Combinatorial Algorithms for Integrated Circuit.* Wiley, 1990.

[154] L. A. Levin. Universal sequential search problems. *Problemy Peredachi Informatskii*, 9(3):115–116, 1973.

[155] Manuel López-Ibáñez, J'rémie Dubois-Lacoste, Thomas Stẗzle Leslie Pérez-Cáceres, and Mauro Birattari. The irace package: Iterated racing for automatic algorithm configuration. 2016.

[156] H. R. Lourenço, O. Martin, and T. Stützle. A beginner's introduction to iterated local search. In *Proceedings of MIC'2001. Metaheuristics International Conference. Porto. Portugal*, volume 1, 2001.

[157] H. R. Lourenço, O. Martin, and T. Stützle. Iterated local search. In *Handbook of Metaheuristics. F. Glover and G. Kochenberger, Eds. International Series in Operations Research and Management Service. Kluwer Academic Publishers, Norwell, MA*, volume 57, pages 321–353, 2002.

[158] J. F. Lynch. The equivalence of theorem proving and the interconnection problem. *ACM SIGDA*, Newsletter 5, 1975.

[159] B. Ma and L. Wang. On the inapproximability of disjoint paths and minimum steiner forest with bandwidth constrainst. *Journal of Computer and System Sciences*, 60 (1), 2000.

[160] V. Maniezzo. Exact and approximate nondeterministic tree-search procedures for the quadratic assignment problem. *INFORMS Journal on Computing*, 11(4):358–369, 1999.

[161] A. Marino, A. Prügel-Bennett, and C. A. Glass. Improving graph colouring with linear programming and genetic algorithms. In *Proceedings of EUROGEN 99*, pages 113–118, 1999.

[162] Bernardo Martín, Ángel Sanchez, César Beltrán-Royo, and Abraham Duarte. A matheuristic approach for solving the edge-disjoint paths problem. *Sixth International Workshop on Model-based Metaheuristics (Matheuristics'16), Brussels, Belgium*, 2016.

[163] Bernardo Martín, Ángel Sanchez, and Abraham Duarte. Resolución del problema de los caminos disjuntos en grafos dirigidos usando técnicas metaheurísticas. *I Jornadas sobre algoritmos evolutivos y metaheurísticas (JAEM 2007). Congreso español de informática*, 2007.

[164] O. Martin, S. W. Otto, and E. W. Felten. Large-step markov chains for the traveling salesman problem. *Complex Syst.*, 5, 3:299–326, 1991.

[165] O. Martin, S. W. Otto, and E. W. Felten. Combining simulated annealing with local search heuristics. *Annals of Operations Research*, 63:57–75, 1996.

[166] C. A. Martínez. *Metaheurísticas híbridas aplicadas al Problema de Ruteo de Arcos Capacitados*. PhD thesis, Departamento de Computación. Universidad de Buenos Aires, 2011.

[167] Ernesto Q. V. Martins and Marta M. B. Pascoal. A new implementation of yens ranking loopless paths algorithm. *Quarterly Journal of the Belgian, French and Italian Operations Research Societies*, 1 (2):121–133, 2002.

[168] Ernesto Q. V. Martins, Marta M. B. Pascoal, and José Santos. A new improvement for a k shortest paths algorithm. *Invest Oper*, 21, 2000.

[169] Ernesto Q. V. Martins, Marta M. B. Pascoal, and José L. E. Santos. The k shortest paths problem. Technical report, Departamento de Matemática, Universidad de Coimbra, Portugal, 1998.

[170] D. Marx. Eulerian disjoint paths problem in grid graphs is np-complete. *Discrete Applied Mathematics*, 2004.

[171] T. Mautor and P. Michelon. Mimausa: A new hybrid method combining exact solution and local search. *Second International Conference on Meta-Heuristics (MIC97)*, 1997.

[172] Alberto Medina, Anukool Lakhina, Ibrahim Matta, and John Byers. Brite: An approach to universal topology generation. In *Proceedings of the International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunications Systems- MASCOTS'01, Cincinnati, Ohio*, 2001.

[173] Alberto Medina, Anukool Lakhina, Ibrahim Matta, and John Byers. Brite: Universal topology generation from a users perspective. Technical report, Computer Science Department, Boston University, 2001.

[174] Mark M. Meerschaert. *Mathematical modeling.* Academic Press, Inc., 2007.

[175] Kurt Mehlhorn and Peter Sanders. *Algorithms and Data Structures: The Basic Toolbox.* Springer, 2008.

[176] N. Melab, E. G. Talbi, M. S. Mezmaz, and B.Wei. Handbook of bioinspired algorithms and applications, chapter 36. parallel hybrid multi-objective meta-heuristics on p2p systems. *Chapman and Hall/CRC, Taylor and Francis*, pages 649–663, 2006.

[177] M. Mezard and A. Montanari. *Information, physics, and computation.* Oxford University Press., 2009.

[178] M. Mezard and G. Parisi. The cavity method at zero temperature. *J. Stat. Phys.*, 111(1-2):1–34, 2003.

[179] M. Mezard, G. Parisi, and R. Zecchina. Analytic and algorithmic solution of random satisfability problems. *Science*, 297 (5582), 2002.

[180] M. Middendorf and F. Pfeiffer. On the complexity of the disjoint path problem. *Combinatorica*, 13 (1):97–107, 1993.

[181] G. L. Nemhauser, M. W. P. Savelsbergh, and G. C. Sigismondi. Minto, a mixed integer optimizer. *Operations Research Letters*, 15(1):47–58, 1994.

[182] S. Nesmachnow. Evaluating simple metaheuristics for the generalizad steiner problem. *Journal of Computer Science and Technology*, 5 (5), 2016.

[183] Isaac Newton and Naomi Campbell. A re-formulation of gravity with respect to really cool models. *Jornal of Funny Physics*, 35:39–78, 1997.

[184] T. Nishizeki, J. Vygen, and X. Zhou. The edige disjoint paths problem is np-complete for series-paralell graphs. *Discrete Applied Mathematics*, 2001.

[185] V. Nwana, K. Darby-Dowman, and G. Mitra. A co-operative parallel heuristic for mixed zero-one linear programming: Combining simulated annealing with branch and bound. *European Journal of Operational Research*, 164:12–23, 2005.

[186] M. Palpant, C. Artigues, and P. Michelon. Lssper: Solving the resource-constrained project scheduling problem with large neighbourhood search. *Annals of Operations Research*, 31 (1):237–257, 2004.

[187] Millie Pant, Radha Thangaraj, and V. P. Singh. Particle swarm optimization with crossover operator and its engineering applications. *IAENG International Journal of Computer Science*, 2, 2009.

[188] C. H. Papadimitriou and K. Steiglitz. *Combinatorial optimization: algorithms and complexity*. Prentice-Hall, Inc., 1998.

[189] M. Pascoal and E. Martins. An implementation of k-shortest path algorithm. https://code.google.com/archive/p/k-shortest-paths/.

[190] A. Plateau, D. Tachat, and P. Tolla. A hybrid search combining interior point methods and metaheuristics for 0-1 programming. *International Transactions in Operational Research*, 9:731–746, 2002.

[191] S. Plotkin. Competitive routing in atm networks. *IEEE J. Selected Areas in Communications*, 1995.

[192] G. Pólya. *How to Solve It*. Princeton University Press, 1957.

[193] M.C. Portmann, A. Vignier, D. Dardihac, and D. Dezalay. Branch and bound crossed with ga to solve hybrid flowshops. *European Journal of Operational Research*, 107 (2):389–400, 1998.

[194] J. Puchinger and G. R. Raidl. An evolutionary algorithm for column generation in integer programming: an effective approach for 2d bin packing. In *X. Yao et al., editors, Parallel Problem Solving from Nature PPSN VIII*, volume 3242 of LNCS, pages 642–651, 2004.

[195] J. Puchinger and G. R. Raidl. Models and algorithms for three-stage twodimensional bin packing. Technical report, TR 18610404, Institute of Computer Graphics and Algorithms, Vienna University of Technology, submitted to the European Journal of Operations Research, 2004.

[196] J. Puchinger and G. R. Raidl. Combining metaheuristics and exact algorithms in combinatorial optimization: a survey and classification. In *J. Mira and J.R. Álvarez (Eds.): IWINAC 2005. Springer-Verlag Berlin Heidelberg*, volume LNCS 3562, pages 41–53, 2005.

[197] J. Puchinger, G. R. Raidl, and G. Koller. Solving a real-world glass cutting problem. In *J. Gottlieb and G. R. Raidl, editors, Evolutionary Computation in Combinatorial Optimization EvoCOP 2004*, volume 3004 of LNCS, pages 162–173, 2004.

[198] P. Raghavan and E. Upfal. Efficient all-optical routing. *Annual ACM Symposium on Theory of Computing*, 1995.

[199] G. R. Raidl. An improved genetic algorithm for the multiconstrained 01 knapsack problem. In *D. B. Fogel, editor, Proceedings of the 1998 IEEE International Conference on Evolutionary Computation. IEEE Press*, pages 207–211, 1998.

[200] G. R. Raidl and H. Feltl. An improved hybrid genetic algorithm for the generalized assignment problem. In *Proceedings of the 2003 ACM Symposium on Applied Computing*, pages 990–995, 2004.

[201] Sixto Ríos. *Investigación Operativa. Programación lineal y aplicaciones.* Centro de Estudios Ramón Areces S. A., 1996.

[202] N. Robertson and P. Seymour. Graph minors. xiii. the disjoint path problem. *Journal of Combinatorial Theory Series B*, 63:65–110, 1995.

[203] P. Saavedra. Introducción a la optimización no lineal. Technical report, Departamento de Matemáticas. Universidad Autónoma Metropolitana, Iztapalapa, México, 2012.

[204] A. Schrijver. Complexity of disjoint paths problems in planar graphs. *Algorithms - ESA '93: First Annual European Symposiums*, 1993.

[205] P. Seymour. Disjoint paths in graphs. *Discrete Math*, 29, 1980.

[206] P. Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In *Principles and Practice of Constraint Programming CP98, 4th International Conference, Springer-Verlag, Pisa, Italy*, volume 1520 of LNCS, pages 417–431, 1998.

[207] Y. Shiloach. A polynomial solution to the undirected path problem. *ACM*, 27, 1980.

[208] E. A. Silver and R. V. V. Vidal. A tutorial on heuristic methods. *European Journal of Operations Research*, 1980.

[209] A. Srinivasan. Improved approximations for edge-disjoint paths, unspplitable flow and related routing problems. *38th Annual IEEE Symposium on Foundations of Computer Science (FOCS'97). IEEE Computer Society Press*, 1997.

[210] A. T. Staggemeier, A. R. Clark, U. Aickelin, and J. Smith. A hybrid genetic algorithm to solve a lot-sizing and scheduling problem. In *Proceedings of the 16th triannual Conference of the International Federation of Operational Research Societies, Edinburgh, U.K.*, 2002.

[211] T. Stützle. Iterated local search for the quadratic assignment problem. Technical report, FG Intellektik, TU Darmstadt, 1999.

[212] T. Stützle. Local search algorithms for combinational problems. analysis, algorithms and new applications. *DISKI. Dissertationen zur Künstliken Intelligenz. infix, Sankt Augustin, Germany*, 1999.

[213] Van Luong T., Taillard E., Melab N., and Talbi E.G. Parallelization strategies for hybrid metaheuristics using a single gpu and multi-core resources. In *Coello C.A.C., Cutello V., Deb K., Forrest S., Nicosia G., Pavone M. (eds) Parallel Problem Solving from Nature - PPSN XII. PPSN 2012. Lecture Notes in Computer Science, vol 7492. Springer, Berlin, Heidelberg*, 2012.

[214] E. G. Talbi. A taxonomy of hybrid metaheuristics. *Journal of Heuristics. Kluwer Academic Publishers*, 8:541–564, 2002.

[215] S. Talukdar, L. Baeretzen, A. Gove, and P. de Souza. Asynchronous teams: Cooperation schemes for autonomous agents. *Journal of Heuristics*, 4:295–321, 1998.

[216] S. Talukdar, S. Murty, and R. Akkiraju. Asynchronous teams. *Handbook of Metaheuristics. International Series in Operations Research & Management Science. Kluwer Academic Publishers*, 57:537–556, 2003.

[217] H. Tamura, A. Hirahara, I. Hatono, and M. Umano. An approximate solution method for combinatorial optimisation. *Transactions of the Society of Instrument and Control Engineers*, 130:329–336, 1994.

[218] T. Thomassen. 2-linked graphs. *Europ. J. Combin.*, 1980.

[219] P. Thompson and J. Orlin. The theory of cycle transfers. Technical report, MIT Operations Research Center, Boston, MA, 1989.

[220] P. Thompson and H. Psaraftis. Cycle transfer algorithm for multivehicle routing and scheduling problems. *Operations Research*, 41:935–946, 1993.

[221] Hideo Umeki. *The geometry package*, July 2002.

[222] Oxford University. *Oxford English Dictionary.* Oxford University Press, 1989.

[223] M. Vasquez and J-K. Hao. A hybrid approach for the 0-1 multidimensional knapsack problem. *Proceedings of the International Joint Conference on Artificial Intelligence 2001*, pages 328–333, 2001.

[224] J. Vygen. Disjoint paths. Technical report, 94816, Research Institute for Discrete Mathematics, University of Bonn, 1995.

[225] J. Vygen. Np-completeness of some edge-disjoint paths problems. *Discrete Applied Mathematics*, 61 (1), 1995.

[226] K. Weihe. Edge-disjoint (s-t)-paths in undirected planar graphs in linear time. *Journal of Algorithms*, 23 (1), 1997.

[227] D. H. Wolpert and W. G. Macready. No free lunch theorems for search. Technical report, Santa Fe Institute, 1995.

[228] D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1 (67), 1997.

[229] D. L. Woodruff. A chunking based selection strategy for integrating meta-heuristics with branch and bound. In *S. Voss et al., editors, Metaheuristics: Advances and Trends in Local Search Paradigms for Optimization, pages 499511. Kluwer Academic Publishers*, 1999.

[230] Grupo Análisis Decisiones y Estadística. Búsqueda inteligente basada en metaheurísticas: Colonia de hormigas. *Departamento Inteligencia Artificial. Facultad de Informática. Universidad Politécnica de Madrid.*

[231] M. Dorigo y T. Stützle. *Ant Colony Optimization.* The MIT Press, 2004.

[232] M. Yaghini. Metaheuristic development. 2010.

[233] J. Y. Yen. An algorithm for
finding shortest routes from all source nodes to a given destination in general networks. *Quarterly of Applied Mathematics*, 27:526–530, 1970.

[234] S. H. Zanakis and J. R. Evans. Heuristic optimization: why, when and how to use it. *Interfaces*, 11, 1981.