UNIVERSIDAD DE GRANADA

Resolución del Problema Militar de Búsqueda de Camino Óptimo Multiobjetivo mediante el uso de Algoritmos de Optimización basados en Colonias de Hormigas

TESIS DOCTORAL

Antonio Miguel Mora García

Granada, 2009

Departamento de Arquitectura y Tecnología de Computadores

Editor: Editorial de la Universidad de Granada
Autor: Antonio Miguel Mora García
D.L.: En trámite
ISBN: En trámite

UNIVERSIDAD DE GRANADA

Resolución del Problema Militar de Búsqueda de Camino Óptimo Multiobjetivo mediante el uso de Algoritmos de Optimización basados en Colonias de Hormigas

Memoria presentada por

Antonio Miguel Mora García

Para optar al grado de

DOCTOR EN INFORMÁTICA

Fdo. Antonio Miguel Mora García

D. Juan Julián Merelo Guervós, Profesor Titular de Universidad y **D. Pedro A. Castillo Valdivieso**, Profesor Titular de Universidad, del Departamento de Arquitectura y Tecnología de Computadores de la Universidad de Granada

CERTIFICAN

Que la memoria titulada: *"Resolución del Problema Militar de Búsqueda de Camino Óptimo Multiobjetivo mediante el uso de Algoritmos de Optimización basados en Colonias de Hormigas"* ha sido realizada por **D. Antonio Miguel Mora García** bajo nuestra dirección en el Departamento de Arquitectura y Tecnología de Computadores de la Universidad de Granada para optar al grado de Doctor en Informática.

Granada, a 23 de Marzo de 2009

Fdo. Juan Julián Merelo Guervós
Director de la Tesis

Fdo. Pedro A. Castillo Valdivieso
Director de la Tesis

A Clara, mi familia y amigos,

Agradecimientos

Quisiera dedicar esta tesis a Clara por su paciencia conmigo sobretodo en las últimas semanas que he pasado enclaustrado en el sótano de nuestra casa.

También lo dedico a mi familia, especialmente a mis padres y mi hermano, por el interés puesto en mi salud y mis niveles de sueño y agobio durante la elaboración de la misma.

Además quisiera agradecer a mis compañeros de trabajo (Luis Javier, Juanlu, Pablo (Fergu), Maribel, Gustavo, Héctor, Manolo, Jesús, Alberto, José Luis, Eduardo, Paco, ...), y en general a todo el Departamento de ATC por su apoyo e interés durante el desarrollo de mi trabajo.

Tampoco quisiera olvidarme de los compañeros del MADOC que me incluyeron en el proyecto SIMAUTAVA primeramente y me prestaron su ayuda al término de éste, en especial a Cristian y Juán.

Quisiera expresar de forma especial mis agradecimientos a J.J. Merelo y Pedro Castillo por su dirección, consejos y ayuda que han conseguido que esta tesis llegue a buen puerto.

Por último, quisiera dar las gracias a J.J. por haberme dado la oportunidad de trabajar en este departamento y desarrollar esta tesis.

Resumen

En este trabajo se presentan una serie de algoritmos desarrollados para la resolución de un problema de búsqueda de camino óptimo, atendiendo a varios criterios y dentro un entorno que modela un campo de batalla militar recorrido por una compañía. Dichos algoritmos han sido planteados como algoritmos de optimización basada en colonias de hormigas para la resolución de problemas multi-objetivo (OCHMO), centrados en la búsqueda de soluciones para problemas con distinto número de objetivos.

Los algoritmos propuestos han sido diseñados para trabajar con uno, dos o cuatro objetivos y se ha postulado un algoritmo para tratar cualquier número de objetivos. Todos han sido bautizados a partir de las siglas CHAC (Compañía de Hormigas ACorazadas), que relacionan los algoritmos de OCH con el entorno militar y con la unidad que se considera.

Además de estos algoritmos, también han sido estudiados y adaptados a la resolución del problema una serie de métodos propuestos en la bibliografía para la resolución de otros problemas. Estos métodos han servido como base comparativa con los algoritmos originales propuestos en la tesis.

Se han modelado para su resolución varios escenarios, partiendo de campos de batalla del juego Panzer General©y definiendo las propiedades y restricciones necesarias para hacerlos fieles a la realidad. Dichos escenarios han sido resueltos aplicando tanto los algoritmos propuestos, como los adaptados.

Los resultados obtenidos demuestran que los algoritmos propuestos ofrecen muy buenas soluciones, mejores en todos los casos que las obtenidas con los métodos rediseñados de la bibliografía. Lo que ha permitido definir varios métodos para solucionar problemas de este tipo en base a su número de objetivos.

Índice general

1. Introducción **1**
1.1. Optimización basada en Colonias de Hormigas 3
1.2. Problemas multiobjetivo 4
1.3. Estructura general de la tesis 5

2. Definición formal del problema **7**
2.1. Problema general de búsqueda de camino óptimo 7
2.1.1. Definición general 7
2.1.2. Problema de búsqueda de camino óptimo multiobjetivo 11
2.1.3. Principales algoritmos para la resolución de problemas de búsqueda de camino óptimo 13
2.2. El problema militar de búsqueda de camino óptimo multiobjetivo 15
2.2.1. Establecimiento del problema a resolver 15
2.2.2. El problema en un simulador 17
2.2.2.1. Justificación del simulador 17
2.2.2.2. Modelado del problema 18

3. Optimización basada en Colonias de Hormigas **24**
3.1. Introducción 24
3.2. Hormigas como agentes artificiales de computación 26
3.2.1. Hormigas naturales y hormigas artificiales 26
3.2.2. Problemas resolubles con OCH 28
3.2.3. La hormiga artificial 28
3.3. Estructura general de un algoritmo OCH 30
3.4. Modelos principales 33
3.4.1. El Sistema de Hormigas 33
3.4.2. El Sistema de Colonia de Hormigas 35
3.4.3. El Sistema de Hormigas Max-Min 37

3.4.4. El Sistema de Hormigas con Ordenación 38
3.5. Aplicaciones de los algoritmos OCH 39

4. Problemas y algoritmos multiobjetivo 41
4.1. Definición de un problema multiobjetivo 42
4.2. Algoritmos multiobjetivo 44
4.2.1. Algoritmos evolutivos multiobjetivo 45
4.2.2. Algoritmos de Optimización basada en Colonias de Hormigas multiobjetivo 47

5. Algoritmos estudiados y desarrollados para resolver el problema 50
5.1. Algoritmo CHAC 51
5.1.1. Descripción detallada del algoritmo 53
5.1.1.1. Funciones heurísticas 54
5.1.1.2. Reglas de transición de estados 55
5.1.1.3. Funciones de evaluación 58
5.1.1.4. Actualización de feromona 59
5.1.2. Pseudocódigo 60
5.2. Algoritmo mono-CHAC 60
5.2.1. Descripción detallada del algoritmo 61
5.2.1.1. Función heurística 63
5.2.1.2. Regla de transición de estados 64
5.2.1.3. Función de evaluación 65
5.2.1.4. Actualización de feromona 65
5.2.2. Pseudocódigo 66
5.3. Algoritmo CHAC-4 66
5.3.1. Descripción detallada del algoritmo 69
5.3.1.1. Funciones heurísticas 70
5.3.1.2. Reglas de transición de estados 71
5.3.1.3. Funciones de evaluación 75
5.3.1.4. Actualización de feromona 75
5.3.2. Pseudocódigo 77
5.4. Algoritmo CHAC-N 78
5.4.1. Descripción detallada del algoritmo 80
5.4.1.1. Funciones heurísticas 80
5.4.1.2. Reglas generales de transición de estados . . . 81
5.4.1.3. Funciones de evaluación 83

5.4.1.4. Actualización de feromona 84
5.4.2. Pseudocódigo 85

6. Algoritmos de la bibliografía adaptados para resolver el problema 87
6.1. Algoritmo MOACS 88
6.2. Algoritmo BiCriterion Ant 91
6.3. Algoritmo Greedy 97

7. Resolución de escenarios reales. Análisis y justificación de resultados 101
7.1. Definición de escenarios a resolver 102
7.1.1. Mapa Panzer General - Dos Enemigos y Ríos 102
7.1.2. Mapa Panzer General - Montañas 105
7.2. Parámetros para la experimentación 107
7.2.1. Parámetros de los algoritmos 108
7.2.2. Pesos 112
7.2.3. Restricciones o características del problema 114
7.3. Variaciones de los algoritmos en base a la parametrización . . 116
7.3.1. CHAC-extremos 117
7.3.2. CHAC-4-extremos 119
7.4. Resolución de escenarios 119
7.4.1. Consideraciones previas a la resolución de escenarios . 120
7.4.2. Mapa Panzer General - Dos Enemigos y Ríos 123
7.4.3. Mapa Panzer General - Montañas 129
7.5. Conclusiones relativas a los experimentos 132

8. Conclusiones y Principales Aportaciones 147

APÉNDICES 155
A. El Simulador SIMBAD 156
B. Mini-Simulador SIMAUTAVA con grid hexagonal (mSS-HEXA) 157
B1. Introducción 157
B1.1. Restricciones 160
B1.2. Terreno (Celdas) 161
B1.3. Subtipos de Celdas 162
B2. Mapas en el Simulador 162
B2.1. Tipos y Subtipos de Celdas 163

B3. Herramientas de Visualización 165
B3.1. Marcar Celdas Vistas y Ocultas 165
B3.2. Marcar Obstáculos Naturales 165
B3.3. Marcar Área de Capacidad de Adquisición . . 166
B4. Visualización de Soluciones 167
B4.1. Datos de Soluciones 167
B4.2. Descripción de Visualización de Soluciones . . 171
B5. Funcionalidad como Simulador de mSS-HEXA 172

Referencias Bibliográficas **175**

Índice de figuras

2.1. Dimensiones de una celda en base a las dimensiones reales en metros que se pretenden modelar. l es el lado y a es la apotema del hexágono. 19

2.2. Mapa de ejemplo de 30x30 celdas que muestra los tipos de celdas y sus colores correspondientes (marrón-normal, verde-bosque, azul-agua, negro- obstáculo). Se han etiquetado los elementos modelados para identificarlos mejor. Los niveles de color indican la altura de las celdas (colores claros indican profundidad, los oscuros elevación). El punto origen es la celda con borde negro y el destino la de borde amarillo. La unidad enemiga aparece con borde rojo y las celdas alrededor de ella corresponden a la zona de impacto de sus armas. Los colores indican la letalidad de las mismas (cuanto más oscuros, más letales serán). 23

4.1. Ejemplo de Frente de Pareto en un problema con dos funciones objetivo (F1 y F2). Los puntos negros representan las soluciones del Frente de Pareto (a y b entre ellas) de un problema con 2 objetivos (F1 y F2). Las soluciones dominadas aparecen en color gris (por ejemplo c). 43

5.1. Conversión de vecindario de celdas a grafo. Cada celda será un nodo y estará conectada con sus vecinos (celdas que la rodean). Ejemplos en una rejilla cuadrada y otra hexagonal respectivamente. 52

7.1. Ejemplo de modelado de un mapa real en el que se tienen un lago, ríos y mucha vegetación. A la derecha el mapa real y a la izquierda la capa de información inferior. 103

7.2. Mapa real modelado en la Figura 7.1, en el que se han señalado a la derecha las áreas correspondientes a la capacidad de adquisición de los enemigos (celdas con tramas en rojo) y de la unidad (celdas con tramas en amarillo). A la izquierda, se han marcado las zonas vistas y ocultas desde el punto de vista de los enemigos, siendo ocultas las celdas sombreadas. 104

7.3. Mapa de 45x45 en el que se muestra una zona montañosa con varias depresiones y picos. La unidad está situada al suroeste (con borde negro) y el punto de destino al norte (con borde amarillo). La imagen de la derecha corresponde a la capa de información subyacente que modela el mapa realista de la izquierda. 105

7.4. Mapa real modelado en la Figura 7.3, en el que se han señalado a la izquierda, los obstáculos naturales que la unidad no puede atravesar (diferencias de altura mayor de 2 entre celdas vecinas). A la derecha se muestra la zona a valorar cuando se considera la celda marcada con un punto negro (área definida por la capacidad de adquisición). 107

7.5. Resultados obtenidos por el algoritmo CHAC aplicando cada una de las reglas de transición (RTC y RTD), para el mapa MPG-2EneRios. A la izquierda se muestra la solución considerada como el camino más rápido y a la derecha el camino más seguro. Las celdas con borde rosa son ocultas a los enemigos y las de borde negro son vistas por alguno de ellos. 134

7.6. Resultados obtenidos por el algoritmo CHAC-4 aplicando cada una de las reglas de transición (RTC y RTD), para el mapa MPG-2EneRios. A la izquierda se muestra la solución considerada como el camino más rápido y a la derecha el camino más seguro. Las celdas con borde rosa son ocultas a los enemigos y las de borde negro son vistas por alguno de ellos. 135

7.7. Resultados obtenidos por los algoritmos MOACS (arriba) y BiAnt (abajo) para el mapa MPG-2EneRios. A la izquierda se muestra la solución considerada como el camino más rápido y a la derecha el camino más seguro. Las celdas con borde rosa son ocultas a los enemigos y las de borde negro son vistas por alguno de ellos. 136

7.8. Resultados obtenidos por los algoritmos GRMO (arriba) y mono-CHAC (abajo) para el mapa MPG-2EneRios. A la izquierda se muestra la solución considerada como el camino más rápido y a la derecha el camino más seguro. Las celdas con borde rosa son ocultas a los enemigos y las de borde negro son vistas por alguno de ellos. 137

7.9. Resultados obtenidos por el algoritmo CHAC con configuración extrema (extr-CHAC), aplicando cada una de las reglas de transición (RTC y RTD), para el mapa MPG-2EneRios. A la izquierda se muestra la solución considerada como el camino más rápido y a la derecha el camino más seguro. Las celdas con borde rosa son ocultas a los enemigos y las de borde negro son vistas por alguno de ellos. 138

7.10. Resultados obtenidos por el algoritmo CHAC-4 con configuración extrema (extr-CHAC-4), aplicando cada una de las reglas de transición (RTC y RTD), para el mapa MPG-2EneRios. A la izquierda se muestra la solución considerada como el camino más rápido y a la derecha el camino más seguro. Las celdas con borde rosa son ocultas a los enemigos y las de borde negro son vistas por alguno de ellos. 139

7.11. Resultados obtenidos por el algoritmo CHAC aplicando cada una de las reglas de transición (RTC y RTD), para el mapa MPG-Montañas. A la izquierda se muestra la solución considerada como el camino más rápido y a la derecha el camino más seguro. 140

7.12. Resultados obtenidos por el algoritmo CHAC-4 aplicando cada una de las reglas de transición (RTC y RTD), para el mapa MPG-Montañas. A la izquierda se muestra la solución considerada como el camino más rápido y a la derecha el camino más seguro. 141

7.13. Resultados obtenidos por los algoritmos MOACS (arriba) y BiAnt (abajo) para el mapa MPG-Montañas. A la izquierda se muestra la solución considerada como el camino más rápido y a la derecha el camino más seguro. 142
7.14. Resultados obtenidos por los algoritmos GRMO (arriba) y mono-CHAC (abajo) para el mapa MPG-Montañas. En el caso de GRMO, a la izquierda se muestra la solución considerada como el camino más rápido y a la derecha el camino más seguro. 143
7.15. Resultados obtenidos por el algoritmo CHAC con configuración extrema (extr-CHAC), aplicando cada una de las reglas de transición (RTC y RTD), para el mapa MPG-Montañas. A la izquierda se muestra la solución considerada como el camino más rápido y a la derecha el camino más seguro. 144
7.16. Resultados obtenidos por el algoritmo CHAC-4 con configuración extrema (extr-CHAC-4), aplicando cada una de las reglas de transición (RTC y RTD), para el mapa MPG-Montañas. A la izquierda se muestra la solución considerada como el camino más rápido y a la derecha el camino más seguro. . . . 145

A1. Simulador SIMBAD. Cartografía real (zona Asturias) con cuatro unidades del bando azul señaladas por sus símbolos tácticos. 157
A2. Datos asociados a las unidades en simulación, en este caso un batallón de carros de combate, formado por tres compañías y el puesto de control. 158
A3. Algunas acciones asociadas a las unidades, según su tipo, situación y estado. 158
B1. Pantalla inicial de mSS-HEXA (vers. H.1.3) 159
B2. Dimensiones de una celda en base a las dimensiones reales en metros que se pretenden modelar. l es el lado y a es la apotema del hexágono. 160
B3. Colores asociados a los tipos y alturas en mSS-HEXA 164
B4. Apariencia de las celdas en mSS-HEXA según sus subtipos . . 164
B5. Niveles de letalidad y colores asociados a las celdas con impacto de armas en mSS-HEXA 165
B6. Mapa de ejemplos etiquetado para mayor claridad 166
B7. Ejemplo de modelado de un mapa real en el que se tienen un lago, ríos y mucha vegetación. A la derecha el mapa real y a la izquierda la capa de información inferior 167

B8. Ejemplo de marcado de celdas vistas y ocultas (con sombreado) a los enemigos 168
B9. Ejemplo de marcado de obstáculos naturales (líneas negras gruesas) que la unidad no será capaz de atravesar 169
B10. Ejemplo de marcado de capacidad de adquisición para la unidad (sombreado amarillo) y enemigos (sombreado rojo) 170
B11. Ejemplo de una solución marcada sobre el mapa resuelto. Las celdas con borde rosa son ocultas a los enemigos y las de borde negro son vistas por alguno de ellos 171

Índice de Tablas

2.1. Descripción de Salud/Energía y Recursos. Factores que constituyen los puntos de cada tipo y factores que consumen dichos puntos. 21

7.1. Valores de los parámetros utilizados para cada uno de los algoritmos en los experimentos. 110
7.2. Pesos asignados a los términos dentro de las funciones relativas a cada objetivo. Arriba los pesos de las funciones heurísticas y abajo los de las funciones de evaluación. 113
7.3. Valores de los parámetros/restricciones utilizados para cada uno de los algoritmos en los experimentos. 116
7.4. Configuración de parámetros para algoritmo CHAC extremo . 117
7.5. Resultados para el Mapa PG Dos Enemigos y Ríos. (1500 iteraciones, 50 hormigas) 124
7.6. Resultados para el Mapa PG Montañas. (1500 iteraciones, 50 hormigas) 130

B1. Consumos por defecto para salud y recursos según el tipo de celda. 173

Índice de Algoritmos

1. OCH() 31
2. Construir_Solucion(id_hormiga) 32
3. estado_sig Regla_Transicion(estado_act) 32
4. CHAC() 61
5. Construir_Solucion(a) 62
6. mono-CHAC() 66
7. Construir_Solucion_Mono(a) 67
8. CHAC-4() 78
9. Construir_Solucion_4(a) 79
10. CHAC-N() 85
11. Construir_Solucion_N(a) 86
12. MOACS() 92
13. Construir_Solucion_MOACS(a) 93
14. BiAnt() 95
15. Construir_Solucion_BiAnt(a,λ_a) 96
16. GRMO() 99
17. Construir_Solucion_GreedyMO() 100

Acrónimos y Símbolos Más Utilizados en la Presente Memoria

ACO—OCH	Algoritmo de Optimización basada en Colonias de Hormigas (Ant Colony Optimization)
ACS—SCH	Sistema de Colonias de Hormigas (Ant Colony System)
AS—SH	Sistema de Hormigas (Ant System)
AE	Algoritmo Evolutivo
AG	Algoritmo Genético
BiAnt	Algoritmo de hormigas bi-criterio (BiCriterion Ant)
CHAC	Colonia de Hormigas Acorazadas (para problemas bi-criterio)
CHAC-4	Colonia de Hormigas Acorazadas para problemas con 4 objetivos
CHAC-N	Colonia de Hormigas Acorazadas para problemas con N objetivos
CIA	Compañía (Militar)
FP	Frente de Pareto
GRMO	Algoritmo Voraz Multi-Objetivo (Greedy Multi-Objective)
MOACO—OCHMO	Algoritmo de Optimización basada en Colonias de Hormigas Multi-Objetivo (Multi-Objective Ant Colony Optimization algorithm)
MOACS	Sistema de Colonias de Hormigas Multi-objetivo (Multi-Objective Ant Colony System)
MOEA	Algoritmo Evolutivo Multi-Objetivo (Multi-Objective Evolutionary Algorithm)

mono-CHAC	Colonia de Hormigas Acorazadas para problemas con un solo objetivo
MOP—PMO	Problema Multi-Objetivo (Multi-Objective Problem)
PCO	Problema de búsqueda de Camino Óptimo
PCM	Problema de búsqueda de Camino Mínimo
PE	Programación Evolutiva
PG	Programación Genética
STR—RTE—RT	Regla de Transición [de Estados] (State Transition Rule)
RTC	Regla de Transición Combinada
RTD	Regla de Transición basada en Dominancia
TSP	Problema del Viajante de Comercio (Travelling Salesman Problem)
VRP	Problema de Enrutamiento de Vehículos (Vehicle Routing Problem)
VRP-TW	Problema de Enrutamiento de Vehículos con Ventana de Tiempo (Vehicle Routing Problem with Time Window)

---

r	subíndice referido al objetivo de rapidez (minimización de recursos consumidos)
s	subíndice referido al objetivo de seguridad (minimización de salud consumida)
d	subíndice referido al objetivo de la distancia al destino (minimización de la distancia media al destino)
v	subíndice referido al objetivo de la visibilidad (minimización de visibilidad de la unidad por los enemigos)
F_r	Función de evaluación que da prioridad a obtener un camino rápido (menor consumo de recursos)
F_s	Función de evaluación que da prioridad a obtener un camino seguro (menor consumo de salud)

λ	Parámetro de ponderación o priorización de objetivos
λ_k	Parámetro de priorización de objetivos para la hormiga k
α	Parámetro de ponderación de la feromona (OCH general)
β	Parámetro de ponderación de la heurística (OCH general)
$\tau(i,j)$	Feromona en el arco entre los nodos i y j (OCH general)
τ_x	Feromona asociada al objetivo x
τ_0	Nivel de feromona inicial (OCH general)
$\eta(i,j)$	Valor de la función Heurística entre los nodos i y j (OCH general)
η_x	Heurística asociada al objetivo x
L^k	Lista de nodos visitados por la hormiga k
N_i^k	Vecindario alcanzable por la hormiga k, desde el nodo i
ρ	Tasa de evaporación de feromona (OCH general)
q_0	Factor de proporción (SCH)

ω_f^x	Peso asociado al término x en la función f
$R(i,j)$	Consumo de recursos al pasar del nodo i al nodo j
$S(i,j)$	Consumo de salud al pasar del nodo i al nodo j
$d(i,j)$	Distancia (euclídea) entre el nodo i y el nodo j
$O(j)$	Nivel de Ocultación del nodo (celda) j
C_x	Función x de coste para valorar la dominancia
$P(i,j)$	Probabilidad de pasar del nodo i al nodo j (RTE de OCH)
$D(i,j,u)$	Función de dominancia entre los arcos (i,j) y (i,u)
MAX_R	Nivel máximo de recursos que requiere atravesar un nodo (celda)
MAX_S	Nivel máximo de salud que requiere atravesar un nodo (celda)

Capítulo 1

Introducción

Dentro del campo de las ciencias de la computación, se habla de *metaheurísticas* al referirse a técnicas de propósito general para guiar la construcción de soluciones o la búsqueda local en las distintas heurísticas e intentar conseguir una mejora adicional en la calidad de las soluciones. Las metaheurísticas incorporan conceptos de muchos y diversos campos como la genética, la biología, la inteligencia artificial, las matemáticas, la física y la neurología entre otras. Se puede ver una presentación de las más utilizadas en [1].

Algunos ejemplos de metaheurísticas con inspiración en la naturaleza son: *Enfriamiento simulado* [2] (simulated annealing en inglés. Inspirado en el proceso de recocido del acero, en el cual éste se calienta y luego se enfría controladamente, durante dicho enfriamiento muchos de sus átomos se recolocan en posiciones según un pequeño componente aleatorio, obteniendo una mejor configuración final de la que se tenía antes del calentamiento. La metaheurística se basa en generar soluciones vecinas a la que se tiene en cada paso y aceptarlas en base a un parámetro llamado temperatura (T), con un pequeño componente aleatorio). *Inteligencia de enjambre* [3] (swarm intelligence en inglés. Inspirada en el comportamiento de los sistemas naturales con múltiples agentes, como las colonias de hormigas, las bandadas de pájaros, los bancos de peces, etc. Se considerarán un conjunto de agentes simples que interaccionarán entre sí y con el medio en el que se encuentren para conseguir un comportamiento emergente que ayude a solucionar el problema planteado). *Algoritmos evolutivos* [4] (genetic/evolutionary algorithms en inglés. Inspirados en el proceso de evolución genética que se da en la naturaleza. Modelan las posibles soluciones al problema a resolver co-

mo individuos y aplican de manera iterativa un proceso de selección de los mejores, reproducción para crear nuevos individuos (hijos) y una pequeña mutación aleatoria).

En esta tesis se estudian y desarrollan algoritmos bioinspirados basados en algunas de estas metaheurísticas, pero adaptados para abordar un problema específico. En concreto se resuelve un problema de búsqueda de camino óptimo atendiendo a varios criterios y dentro un entorno muy particular.

Dichos algoritmos están enmarcados concretamente dentro de la metaheurística de **Optimización basada en Colonias de Hormigas** (OCH) [5] y más generalmente, y dado que se considerarán múltiples criterios en la búsqueda del mejor camino, se engloban dentro de los algoritmos para resolución de **Problemas Multiobjetivo** (PMO) [6].

El problema está tipificado como un **problema de búsqueda de camino óptimo dentro de un grafo**, pero presenta varias peculiaridades y/o restricciones que lo convierten en un tipo de problema más Específico. En concreto, se trata de buscar el mejor camino entre dos puntos, teniendo que, el entorno en el que se deberá encontrar la solución es un *campo de batalla militar*. Esto hace que aparezcan condiciones, restricciones y características a tener en cuenta en la búsqueda que lo diferencian de otros problemas de búsqueda de camino óptimo, como el hecho de considerar el que una unidad militar sea la que deba recorrer dicho camino una vez definido. Además, dicho problema estará planteado para la satisfacción de dos criterios: la maximización de la rapidez al recorrer el camino, y la maximización de la seguridad en el mismo.

Los algoritmos estudiados son diferentes variantes del que hemos dado a llamar **CHAC** (**C**ompañía de **H**ormigas **AC**orazadas), pues se trata de algoritmos basados en colonias de hormigas adaptados para la resolución de un problema dentro de un marco militar. Además, se han planteado como base comparativa, varios algoritmos resultantes de la adaptación de métodos presentados en la bibliografía a la resolución de este problema dentro del mismo entorno.

En el desarrollo de esta tesis, los algoritmos han sido aplicados a la resolución de varios problemas, tanto reales, como sintéticos y comparados, a su vez, con las técnicas descritas en la bibliografía y adaptadas a la resolución del problema, como ya se ha comentado. Los resultados obtenidos muestran la capacidad de los algoritmos diseñados, y la validez de éstos en la resolución de problemas de búsqueda de camino óptimo.

1.1. Optimización basada en Colonias de Hormigas

La Optimización basada en Colonias de Hormigas (OCH) es una metodología heurística y parametrizada (metaheurística) inspirada en el las conclusiones alcanzadas por Pierre-Paul Grassé [7, 8, 9] y Denebourg et al. [10, 11, 12] acerca del comportamiento de algunas especies de hormigas que son capaces de encontrar el camino más corto entre su nido y una fuente de comida, mediante cooperación y en un corto periodo de tiempo. Dicho método está basado en el concepto de *estigmergia* [13], que significa que hay una comunicación entre agentes a través del medio en el que se encuentran. En este caso cada hormiga, mientras camina, deposita en el suelo una sustancia llamada *feromona* que las demás pueden detectar (oler). Cada una de ellas tiende a seguir (ante una bifurcación con varias posibilidades) las concentraciones más altas de dicha sustancia (la cual se evapora tras cierto tiempo), y a su vez, hace su propio aporte. Esto acaba por constituir rastros o 'caminos' de feromona que marcan el mejor camino (el más corto) entre el nido y una fuente, que suele ser de alimento.

Esta metaheurística fue presentada por Dorigo et al. [14] en 1991. Los algoritmos OCH se inspiran en este comportamiento para resolver problemas de optimización combinatoria y usan una colonia de *hormigas artificiales*, que son agentes computacionales y que se comunican entre ellos usando *feromonas*.

Para que un problema sea resuelto mediante un algoritmo OCH, éste debe ser transformado en un grafo con pesos en sus arcos. En cada iteración, cada hormiga construirá un camino completo (solución) moviéndose por el grafo. Una vez construido dicho camino (o durante su construcción), la hormiga irá depositando un rastro de feromona que, generalmente, será función de la bondad de la solución que esté construyendo o haya construido. Por tanto, dicho rastro será una medida (informativa para las demás) de lo deseable que es seguir el mismo camino que la susodicha hormiga.

De modo que, para moverse en el grafo, cada hormiga manejará dos tipos de información: la mencionada feromona o *información memorística* y cierta *información heurística*, la cual depende del problema y se basa en el aprovechamiento de un conocimiento previo del mismo (cuyo valor no cambia durante la ejecución).

En su movimiento por el grafo, las hormigas elegirán normalmente los

nodos con un mejor valor para las dos informaciones (las cuales se combinan), pero existirá un componente estocástico por el que una hormiga podrá moverse a nodos con valores menores, pues es posible que la solución final obtenida siguiendo esos nodos sea mejor. Se trata pues, de un componente para potenciar la exploración, la cual es fundamental para la resolución de problemas combinatorios complejos.

De esta forma todas las hormigas colaborarán para encontrar la mejor solución para el problema (el mejor camino dentro del grafo), lo que modela un comportamiento emergente global.

Los algoritmos OCH se presentaron en dos modelos inicialmente: el Sistema de Hormigas (SH) [15] y el Sistema de Colonia de Hormigas (SCH) [16]. Pero actualmente hay muchas variantes y nuevos métodos.

Esta metaheurística, así como sus principales modelos y variantes será comentada más extensamente en el Capítulo 3.

1.2. Problemas multiobjetivo

Los problemas de optimización multiobjetivo (o multicriterio) [6] son aquellos en los que varios objetivos deben ser optimizados simultáneamente. Dichos objetivos serán totalmente independientes, por lo que la mejora (o empeoramiento) en uno de ellos no afectará al resto. De modo que, generalmente, no habrá una única solución óptima para el problema, ya que cada solución podrá ser buena respecto a uno o varios objetivos, pero muy mala respecto a otros, como de hecho pasa habitualmente (usualmente la mejora de la solución respecto a un objetivo se traduce en un empeoramiento respecto a otro).

Formalmente, la resolución de un problema multiobjetivo consiste en maximizar o minimizar una función *f(x)* compuesta por k funciones de coste (una por objetivo) y considerando n parámetros (variables de decisión):

$$\begin{aligned} &f(x) = (C_1(x), C_2(x), ..., C_k(x)) \\ &x = (x_1, x_2, ..., x_n) \ \in \ X \end{aligned} \tag{1.1}$$

En un problema de optimización multiobjetivo típico, habrá un conjunto de soluciones que serán mejores que el resto considerando todos los objetivos, dicho conjunto es conocido como *Frente de Pareto* o *Conjunto de Pareto*.

La creación de este conjunto está relacionada con el concepto de *dominancia*, definido como (a domina b, considerando la minimización de los

objetivos):

$$\begin{array}{l} a \prec b \;\; if: \\ \forall i \in 1,2,...,k \mid C_i(a) \leq C_i(b) \;\; \wedge \;\; \exists j \in 1,2,...k \mid C_j(a) < C_j(b) \end{array} \tag{1.2}$$

Dónde $a \in X$ y $b \in X$ son dos soluciones (vectores de decisión) con n variables, y cada C es la función de coste asociada a cada objetivo.

De este modo, las soluciones incluidas en el Frente de Pareto son conocidas como soluciones *no dominadas* y el resto de las soluciones posibles son llamadas soluciones *dominadas*. Cualquier solución no dominada será considerada como una solución igual de aceptable para el problema.

Los algoritmos adaptados e implementados para la resolución de problemas multiobjetivo son los conocidos como *algoritmos multiobjetivo*.

En el Capítulo 4 se expondrá más formalmente la definición de este tipo de problemas y se comentarán los principales algoritmos multiobjetivo.

1.3. Estructura general de la tesis

Tras el Capítulo de introducción general de la tesis, el Capítulo 2 describe de manera formal el problema que se abordará en la misma, partiendo de su formulación general (tanto mono, como multiobjetivo) y detallando posteriormente las particularidades del caso a estudiar.

Tras esto, se tienen una serie de capítulos dedicados a presentar los conceptos fundamentales en los que se basa este trabajo. De modo que en el Capítulo 3, se describen detalladamente los algoritmos de optimización basada en colonias de hormigas, desde su bioinspiración, pasando por sus bases fundamentales, los modelos principales y algunas de sus aplicaciones.

Posteriormente, en el Capítulo 4 se detallan las propiedades de los problemas multiobjetivo y se presentan los algoritmos más conocidos para resolver este tipo de problemas, desde los algoritmos evolutivos clásicos, hasta los algoritmos de optimización basada en colonias de hormigas para la resolución de este tipo de problemas.

Una vez presentados los aspectos conceptuales de la tesis, el Capítulo 5 está dedicado a describir en profundidad cada uno de los métodos desarrollados para la resolución del susodicho problema.

En el Capítulo 6, se presentan las adaptaciones de algunos algoritmos que aparecen en la bibliografía, para la resolución del mismo problema, a fin de conseguir una base comparativa para con los algoritmos propuestos.

En el Capítulo 7 se aplican los métodos a una serie de problemas (reales y simulados) y se comentan los detalles de los ajustes de parámetros, experimentos y estudios comparativos que se han hecho.

La memoria termina con un Capítulo de conclusiones y líneas de investigación y desarrollo que se sugieren para un futuro.

Capítulo 2

Definición formal del problema

En este apartado se definirá de una manera más rigurosa el problema que se pretende resolver en este estudio. Este problema comparte muchas características de los problemas generales de búsqueda de camino óptimo en grafos, si bien tiene un enfoque multiobjetivo (ya que se tienen en cuenta varios criterios en la búsqueda) y además presenta varias restricciones y características particulares que lo convierten casi en un problema específico.

Parte de esas condiciones y restricciones han sido consideradas a fin de asemejarlo en la medida de lo posible a las características del problema en el mundo real y más específicamente, dentro del entorno militar.

2.1. Problema general de búsqueda de camino óptimo

A continuación se definirá de manera general el problema, describiendo sus propiedades y características, considerando también su definición desde el punto de vista multiobjetivo y enumerando algunos de los ejemplos de problemas de este tipo más conocidos, así como aplicaciones prácticas del mismo (problemas en la realidad que se ajustan a este tipo).

2.1.1. Definición general

El problema que nos ocupa en esta tesis es el llamado **Problema de Búsqueda de Camino Óptimo** PCO (path-finding problem en inglés), el cual se puede definir como: dado un entorno en el que se tienen definidos

distintos puntos y en el que es posible trazar rutas entre ellos, y dados dos puntos cualesquiera dentro de dicho entorno, hallar la ruta entre ambos puntos que mejor que adecue a una serie de criterios predeterminados. Dicha ruta partirá de uno de los puntos, llamado *origen* y llegará hasta el otro, conocido como *destino*, y a su vez, podrá discurrir por varios puntos intermedios.

El PCO más común tiene un único criterio a satisfacer: minimizar la distancia recorrida en el camino; de modo que se tendrá asociada una distancia a la ruta existente entre cada par de puntos en el entorno. La distancia total sería por tanto, la suma de las distancias recorridas durante el movimiento entre cada dos puntos intermedios que se hayan considerado entre el punto origen y el punto destino. Es el conocido como *problema de búsqueda de camino mínimo* o más corto, PCM (shortest path problem, en inglés) [17].

También existen PCOs que consideran ciertas restricciones en la búsqueda, como por ejemplo, el hecho de que el camino discurra por unos puntos determinados, o por el contrario, que dicho camino no pueda atravesar ciertos puntos. Estas restricciones limitan las posibles soluciones a encontrar y determinan la naturaleza del problema, incluso llegando a convertirlo en un caso específico.

Normalmente en la bibliografía solo se hace referencia al PCM, aunque existe una diferencia entre ambos conceptos y es que un PCO podría considerar un criterio a maximizar, por lo que no sería un camino mínimo (ni corto). Del mismo modo, es posible considerar condiciones o restricciones que deba cumplir dicho camino y que no lo conviertan en el más corto según el criterio, pero si en el óptimo.

Se pueden resumir las características de este tipo de problemas de forma esquemática (según lo expuesto en [14]):

- Se pueden tener una serie de restricciones/condiciones del problema.
- Se tiene un conjunto de componentes finitos (puntos).
- Se tiene una serie de estados finitos, donde cada estado corresponde con una secuencia de componentes (lista de puntos recorridos en cada momento).
- Existe una función de vecindad que permite pasar de un estado a otro (hay una serie de puntos a los que se puede pasar en cada momento).
- Cada cambio de estado tiene un coste asociado.

- Una solución es una secuencia de estados que verifica las restricciones/condiciones del problema.
- Hay un coste asociado a cada solución.

Este problema normalmente se resuelve en entornos modelados como *grafos*, por lo que, volviendo a plantearlo su definición sería: dado un grafo y dos nodos (origen y destino), hallar la lista de nodos (o arcos) que se deben atravesar para ir desde el nodo origen al nodo destino considerando una serie de criterios predeterminados. Dicho grafo deberá cumplir una serie de condiciones (según se comenta en [14]):

- Será un grafo con un número finito de nodos.
- Podrá ser dirigido o no.
- Cada nodo está comunicado con, al menos otro mediante un arco que tiene asociado un peso.
- Se considera como solución una secuencia de nodos comunicados mediante arcos (un camino dentro del grafo).
- Cada secuencia solución tiene asociado un coste, que será función de los costes de moverse entre cada dos nodos dentro del camino solución.

Un ejemplo muy conocido de este tipo de problema es el llamado *Problema del Viajante de Comercio*, TSP en adelante (travelling salesman problem en inglés) [18], el cual se puede definir formalmente como: *dado un grafo finito, conexo y ponderado en sus arcos, hallar el circuito hamiltoniano que minimiza la función de coste, siendo esta la suma de los pesos de los arcos.*

Considerando que el peso en los arcos indica la distancia que separa a los nodos que éstos unen, se trataría de un PCM con algunas restricciones y condiciones. El criterio sería obtener el camino de distancia mínima entre origen y destino, siendo ambos puntos el mismo, y además considerando como condición el hecho de que dicha ruta debe pasar por todos los nodos del grafo. Como restricción se podría considerar el que no se permita que el camino pase más de una vez por cada nodo.

Otro ejemplo muy extendido es el del *Problema del Enrutamiento de Vehículos* VRP en adelante (vehicle routing problem, en inglés) [19, 20], el cual se puede definir siguiendo el esquema anterior como: *dado un grafo finito,*

conexo y ponderado en sus arcos y en sus nodos, y dado un nodo determinado, hallar el conjunto de rutas que minimicen la suma de los pesos de los arcos que partan de dicho nodo y lleguen a todos los demás, considerando como condición que el hecho de alcanzar cada nodo implica además un coste relativo al nodo al que se llega, el cual está limitado a priori para cada ruta que parte del nodo elegido.

En este caso, sería más correcto hablar de un PCO que de un PCM, pues lo que se pretende minimizar no es únicamente el coste de recorrer la ruta, sino también el número de rutas necesarias para unir el nodo elegido con todos los demás (condicionadas por la restricción del coste asociado a cada nodo).

Este tipo de problemas han sido adaptados multitud de veces a la resolución de problemas reales, dado que muchos de ellos fueron planteados inicialmente a partir de situaciones que se debían solucionar en la realidad. Como muestra se pueden ver algunos ejemplos, los dos primeros (planteados previamente) toman su nombre de dos de estos casos:

- *TSP o Problema del Viajante de Comercio*: su formulación inicial intentaba encontrar la mejor ruta que debía seguir un comerciante para visitar un conjunto de ciudades (solamente una vez cada una), partiendo de una en concreto y volviendo a la misma. Podía existir o no un camino directo entre dos ciudades. Se buscaba recorrer la mínima distancia total (obtenida como suma de las distancias que separan cada par de ciudades). Por tanto, su resolución se aplicará en empresas que utilicen vendedores comerciales por zonas o por parte de comerciantes ambulantes.

- *VRP o Problema de Enrutamiento de Vehículos*: el cual se planteó como un problema de reparto de artículos desde un almacén a muchos clientes. Por tanto, se consideraban un almacén central, uno o más vehículos y un conjunto de clientes a los que servir. Dichos clientes tenían asignado un número de artículos para recibir y los vehículos una capacidad limitada. Por tanto, el problema era encontrar las rutas más adecuadas (las más cortas y que satisficieran la restricción de la capacidad de los vehículos). La aplicación de este problema será común en empresas transporte a domicilio y logística.

- *Camino mínimo (shortest path)*: se trata de un problema muy común en la realidad y el más intuitivo de los PCM, pues en él se desea en-

contrar la mejor ruta entre dos puntos, considerando las restricciones pertinentes y las distancias entre cada dos puntos a atravesar. Su aplicación a callejeros/mapas interactivos, o a la resolución de escenarios en videojuegos o simuladores está a la orden del día, así como el cálculo de rutas directas que se emplearán en compañías de transportes, policía, bomberos, etc.

Estos problemas han sido ampliamente resueltos en la bibliografía (como se comentará posteriormente en la Sección 2.1.3).

2.1.2. Problema de búsqueda de camino óptimo multiobjetivo

En muchas ocasiones, y dentro de los PCOs, existen varios criterios que debería cumplir una ruta para ser considerada como óptima. Cuando dichos criterios son independientes entre si (es decir, el cumplimiento de uno no implica la satisfacción de otro), se puede hablar de que el problema es multiobjetivo. En el Capítulo 4 se puede ver una descripción de este tipo de problemas.

Por tanto, la definición esquemática del problema que se hizo en el apartado anterior cambiaría levemente, pues:

- En el caso de la definición general del problema:
  - cada cambio de estado tendrá asociados varios costes, uno por cada objetivo que se pretende optimizar en la mejor ruta.
  - cada solución encontrada tendrá asociado un coste por cada uno de los objetivos.
- En el caso de la definición del problema como grafo:
  - cada arco del grafo tendrá asociados varios pesos, uno por cada objetivo a optimizar.
  - cada camino en el grafo propuesto como solución tendrá asociados varios costes, uno por cada uno de los objetivos.

Además, y considerando las propiedades de los problemas multiobjetivo (ver Sección 4.1), por lo general no habrá una única solución óptima (que sea

la mejor para todos los criterios), sino que se tendrán un conjunto de soluciones que serán buenas para ciertos objetivos y peores para otros, pero que serán consideradas como igual de buenas que sus complementarias (buenas para los objetivos para los que éstas eran peores). De forma que habrá múltiples caminos solución a considerar.

De modo que, pensando en el ejemplo general comentado anteriormente, aparte de minimizar la distancia a recorrer, se podría desear optimizar la rapidez de la ruta (la cual puede depender de variables relativas al tipo de ruta), el consumo de recursos que implica dicho camino (que podría estar relacionado con la dificultad de la misma), etc. De la misma forma, se obtendrían varias soluciones, pudiendo ser algunas muy buenas respecto a la distancia recorrida, pero muy malas en cuanto a la rapidez de la ruta o el consumo de recursos que implica, y viceversa. Pero todas ellas serían consideradas como soluciones aceptables.

Respecto a los ejemplos comentados en el apartado anterior, existen variantes multiobjetivo para los mismos, de modo que:

- en el TSP multiobjetivo (TSPMO), se tienen varias funciones de coste, de modo que aparte de la distancia entre los nodos (ciudades), se considera la calidad de la ruta entre ellas, el tiempo requerido para pasar de una a otra o el consumo de combustible empleado.

- en el caso del VRP se define la llamada *ventana de tiempo* (time window en inglés), pasando a llamarse VRPTW. Esta ventana pasa a considerarse parte de un nuevo objetivo, ya que establecerá un intervalo de tiempo en el que será deseable que se acceda a cada uno de los clientes. De este modo, habrá una nueva función de coste que penalizará las rutas que lleguen a un cliente fuera de su ventana de tiempo (antes o después).

- la definición del PCMMO es intuitiva, ya que al igual que en el caso del TSPMO, se añaden nuevas funciones de coste. En este caso, su nombre pasa a tener un significado 'ligeramente incorrecto', debido a que un camino solución, como se ha comentado unas líneas antes, raramente será mínimo para todos los objetivos.

A continuación se enumeran los principales algoritmos empleados para resolver PCOs, tanto con uno como con varios objetivos, y considerando desde los métodos clásicos a las metaheurísticas.

2.1.3. Principales algoritmos para la resolución de problemas de búsqueda de camino óptimo

Existen infinidad de algoritmos en la literatura ideados y desarrollados para la resolución de este tipo de problemas o versiones de los mismos, considerando nuevas o diferentes restricciones, introduciendo componentes dinámicos y un sinfín de variantes. Todos ellos tienen en común el trabajar con problemas modelados como grafos, con las características comentadas en el apartado 2.1.3.

A continuación se hará un breve repaso de los más conocidos o de mayor relevancia en el estado del arte de los PCO (o PCMs).

En primer lugar se debe hacer una distinción entre los métodos de resolución clásicos (generalmente deterministas) y las modernas metaheurísticas (aunque ya cuentan con varias décadas de antigüedad). Dentro de los primeros tenemos como máximos exponentes:

- *Algoritmo de Bellman-Ford* [21]: desarrollado en 1958 como solución al PCM con un único origen dentro de grafos. Este algoritmo construye los caminos más cortos desde el nodo origen a todos los demás nodos del grafo, sirviendo pues para resolver el cálculo del camino mínimo entre el nodo origen y otro cualquiera dentro del grafo. Es posible que en el grafo haya arcos con peso negativo e incluso ciclos (secuencia de arcos que empiezan y terminan en el mismo) con coste negativo.

- *Algoritmo de Dijkstra* [22]: presentado en 1959 para solucionar el mismo problema que el anterior. Toma su nombre por su creador, Edsger Dijkstra. Este algoritmo construye soluciones igualmente desde el origen a todos los demás, y de manera más eficiente, de ahí su mayor fama y utilización. Pero requiere como condición que los pesos asociados a los arcos no sean negativos.

Posteriormente aparecieron los llamados *algoritmos informados*, los cuales consideran información adicional del problema y utilizan *funciones heurísticas* para guiar la búsqueda. Su mayor exponente es:

- *Algoritmo A** [23]: aparecido en 1968 como alternativa a los algoritmos de búsqueda en grafos existentes hasta el momento, los cuales, o bien no consideraban información suplementaria para mejorar la búsqueda (información heurística acerca problema), o si la utilizaban (algoritmos

informados) únicamente consideraban dicha información, lo cual significaba que en la construcción de la solución siempre se incluían los mejores nodos posibles (siguiendo un enfoque *voraz* o greedy en inglés).

A* incorporó la idea de considerar un posible empeoramiento a nivel local (no elegir el mejor nodo) con el fin de obtener una mejor solución global. Para ello tiene en cuenta el coste total de la solución en cada momento. Se trata de un algoritmo de búsqueda en anchura (explora todos los vecinos de cada nodo en cada paso) aunque combina algunos aspectos de la búsqueda en profundidad (se centra en un nodo y explota su camino hasta el final). Es un algoritmo completo, de forma que si hay solución, la encontrará.

En cuanto a las metaheurísticas, existen infinidad de ellas aplicadas a la resolución de PCOs, siendo el TSP el problema más extendido y resuelto mediante gran cantidad de algoritmos (como se puede ver en [24, 25, 26, 27]). Del mismo modo, tanto el VRP [28, 29], como los problemas clásicos de camino mínimo [17], han sido también ampliamente tratados mediante numerosos métodos.

Si bien es cierto que hay una metaheurística que se adapta desde su concepción a la resolución de PCO, los algoritmos de OCH (que se verán extensamente en el Capítulo 3), ya que éstos se idearon expresamente para la resolución de problemas modelados como grafos y cuyas soluciones se pudiesen representar como caminos dentro de dichos grafos. De hecho, el modelo inicial de algoritmo de OCH [30] fue planteado para solucionar el TSP, y del mismo modo, el mismo autor presentó posteriormente una variación del modelo inicial [16] con el mismo fin. Tras la aparición del algoritmo de OCH, hubo muchas variaciones y adaptaciones del mismo centradas en la resolución del TSP como por ejemplo [31, 32, 33, 34]. En cualquier caso, dicho problema se ha convertido en el test que aplican muchos autores a la hora de comprobar si su nueva variación del algoritmo de OCH es competente.

Respecto a la relación entre VRP y OCH, este problema también ha sido muy utilizado en la presentación de nuevos algoritmos [35], tanto adaptados para su resolución, como novedosos y usándolo como problema de test.

Del mismo modo, muchos problemas de camino mínimo han sido resueltos mediante algoritmos de OCH [36, 37].

En los últimos años, los estudios de PCO se están decantando hacia entornos dinámicos, paralelos y sobretodo multiobjetivo, dónde las técnicas clásicas tienen muy poca utilidad, ya que las propiedades de los problemas

multiobjetivo condenan el uso de algoritmos deterministas (éstos únicamente obtienen una solución). Aún así, existen adaptaciones como el A* multiobjetivo (MOA*, en inglés) [38], e incluso una extensión del algoritmo de Dijkstra que considera varios criterios [39]. En dicho entorno, el problema del VRPTW es resuelto por un gran número de metaheurísticas [40, 41], destacando los algoritmos OCHMOs [42, 43, 44, 45]. Del mismo modo, el TSP multiobjetivo, sobretodo en su versión bi-criterio en también resuelto usando distintas metaheurísticas [46, 47, 48] y con asiduidad mediante algoritmos OCHMOs [49], empleándolo nuevamente como problema de test. Para finalizar, el problema de camino mínimo multiobjetivo ha sido también resuelto en diversas ocasiones [50, 51].

2.2. El problema militar de búsqueda de camino óptimo multiobjetivo

Es esta sección, se definirá de forma más específica el problema que se pretende afrontar en este trabajo, justificándolo y situándolo primeramente dentro de un entorno militar y describiendo posteriormente sus condiciones y restricciones, así como los criterios específicos a considerar.

Dichas condiciones y restricciones están muy ligadas al entorno en el que se plantearán y resolverán las diferentes instancias del problema, las cuales serán mapas o escenarios que modelen campos de batalla reales. Es decir, se trabajará con un simulador que incluirá una serie de funciones y características para modelar la realidad de la mejor forma posible.

2.2.1. Establecimiento del problema a resolver

Como se comentó en el Capítulo de Introducción, la definición del problema a resolver, está dentro de los problemas de búsqueda de camino óptimo, presentados en las secciones precedentes. Si bien, éste se encuadra dentro de un entorno de carácter militar, es decir, la búsqueda del mejor camino entre dos puntos se hará dentro de un campo de batalla. La situación en dicho entorno incluirá criterios, condiciones y restricciones que convertirán a este PCO en un problema específico.

La justificación de la resolución de este problema pasa por la elección de un tema de investigación que tuviese una aplicación directa y útil. Considerando entonces el entorno de defensa (militar), dada la gran cantidad

de problemas de toda índole que se afrontan en él [52, 53, 54, 55, 56] y su inmediata aplicabilidad en la mayoría de los casos. Dentro de dicho entorno, el movimiento es la forma de acción fundamental (desde el punto de vista táctico) ya que es la primera parte de la maniobra (movimiento y fuego), que es la acción principal de una unidad. Con él, la unidad se acerca o aleja del enemigo o se desplaza hasta un punto objetivo manteniendo siempre una ventaja táctica (moviéndose por zonas ocultas al él, entrando por un flanco o retaguardia de éste, etc). De forma que, se decidió resolver esta primera acción dentro de la maniobra.

Además, se consideró como objeto de referencia la *Compañía Militar* (CIA), pues es la menor unidad dentro del ejército que puede funcionar, si las condiciones lo requieren, de manera autónoma, sin requerir ningún tipo de control o coordinación con las demás unidades. Esta elección acercará el planteamiento del problema más aún al mundo real, pues en él únicamente se considerará una unidad (obviándose cuestiones inherentes a la presencia de varias unidades en la resolución del problema, como por ejemplo el establecimiento de prioridades o zonas/líneas de acción para coordinarlas).

Otro aspecto a considerar a la hora de afrontar la resolución del problema del movimiento de una CIA militar, es que la elección del mejor itinerario se ve condicionada por dos factores que deben ser sopesados y que obligan a tomar una decisión multicriterio: la **seguridad** y la **rapidez**. La primera prima sobre la segunda cuando se desconoce la situación del enemigo, cuando se busca la sorpresa o en unidades cuya misión es informar. En este caso, el itinerario se trazará buscando zonas ocultas del terreno (no vistas por el enemigo o desde puntos cercanos, caso de no saber dónde está éste) o las más protegidas, lo que conducirá seguramente a itinerarios más largos y fatigosos. La rapidez prima en las misiones de ataque (en las que se cuenta con un tiempo limitado) y en terrenos con poca cobertura. El itinerario más rápido es también el que se recorre en el menor tiempo y el que produce el menor desgaste, pero también es el más expuesto y el que más bajas producirá. Estos criterios son contrapuestos, pero no excluyentes, y deben ser valorados según la misión y la situación táctica. En ambos casos, la elección del mejor itinerario para que la unidad alcance su objetivo es una decisión táctica muy importante.

Muchos de estos conceptos, así como la estructura orgánica del Ejército de Tierra Español, se pueden consultar en la descripción que ofrece la Wikipedia [57].

De forma que, considerando lo anteriormente dicho, el problema inicial

a resolver sería: *la búsqueda (por parte de una CIA) del camino óptimo desde un punto origen hasta un punto destino, dentro del campo de batalla y considerando los criterios de seguridad y rapidez*. Por tanto, en este trabajo se estudiará la resolución del que hemos llamado **problema de camino óptimo bi-criterio de la unidad militar**.

Según lo comentado en las secciones precedentes, se trata de un PCO multiobjetivo, con un único origen y un único destino, y con varias condiciones y restricciones impuestas por la naturaleza del problema (consideraciones tácticas), por las características de la unidad que realiza el movimiento (CIA), y por las propiedades el entorno en el que se desarrolla, pues será un campo de batalla modelado.

Este problema se afrontará haciendo uso de metaheurísticas, por su mejor desempeño en problemas multicriterio y concretamente, se resolverá mediante algoritmos de OCHMO, dados los buenos resultados que han arrojado en muchos otros PCOs (como se comentó en la Sección 2.1.3).

No existen en la bibliografía apenas ejemplos de algoritmos aplicados a la resolución de este tipo de problemas dentro del ámbito militar ([58, 59]) y ninguno considera varios criterios en la búsqueda.

A continuación se describirán las peculiaridades que muestra este problema debidas al entorno en el que se resolverá (campos de batalla modelados), las propiedades de la unidad del problema, y las condiciones inherentes al ámbito militar. Además, se comentarán las posibles restricciones adicionales introducidas para poder afrontar el problema dentro de un entorno lo más realista posible, es decir, un simulador.

2.2.2. El problema en un simulador

A fin de resolver el problema como un modelo, será necesario replantear dicho problema dentro de un entorno que lo haga manejable, esto es, un simulador. Por ello, en la siguiente sección se justificará esta necesidad, al tiempo que en la Sección 2.2.2.2 se enumerarán las propiedades del problema al ser replanteado dentro de dicho entorno.

2.2.2.1. Justificación del simulador

Debido a las características del PCO que se quiere resolver, es necesario realizar una modelización o ajuste de propiedades previos, a fin de poder tratarlo en un computador. Es decir, habrá que definir el contexto de cada

posible problema (campo de batalla particular) como un escenario finito y discreto. Del mismo modo debería ser posible modelar todas las propiedades, condiciones y restricciones inherentes al ámbito en el que se ubica dicho problema, así como a la unidad que se considerará en el mismo. Por tanto, habrá que simular de la manera más fiel posible a la realidad cada uno de estos aspectos en base a los conocimientos que se tienen a priori de cada uno de ellos.

Con ese fin, se define un **simulador**[1], es decir, un entorno en el que se pretende modelar el mundo real (dentro de este ámbito concreto), incorporando todas esas características, así como distintas funcionalidades necesarias tanto para la resolución de un problema, como para la interpretación de las posibles soluciones al mismo. Dicho entorno ofrecerá, por tanto, herramientas para poder plantear escenarios a resolver, es decir, instancias concretas del problema. Del mismo modo, también tendrá utilidades para la valoración, visualización e interpretación de las soluciones asociadas a cada instancia. Tanto unas como otras serán controladas por usuarios humanos. Al mismo tiempo este simulador modelará algunas de las funciones a considerar para la caracterización de un determinado problema y su resolución de la manera más fiel posible a la realidad.

Este entorno ha sido creado durante la realización del presente trabajo, y ha sido bautizado como *mini-Simulador Simautava*[2] (mSS). En el Apéndice B se puede encontrar una breve descripción de sus propiedades y funcionalidad.

En la siguiente sección se describirá el modelado del problema que se ha realizado, comentando las propiedades y restricciones que ello implica.

2.2.2.2. Modelado del problema

La transformación del problema real a tratar en la tesis en un modelo manejable por un simulador, ha implicado varias transformaciones, las cuales se comentan a continuación.

En primer lugar, se ha discretizado **el campo de batalla**, modelándolo como una matriz de celdas hexagonales[3] a fin de conseguir un modelo más

[1] Al tiempo que se desarrollaba este estudio, ya existía un simulador en el MADOC (entidad con la que se colaboraba), el cual se comenta en el Apéndice A, pero al que únicamente tuvimos acceso a nivel educativo

[2] SIMAUTAVA fue el proyecto dentro del cual comenzó el estudio presentado en esta tesis

[3] Aunque en versiones preliminares se modeló considerando una rejilla cuadrada

fiel al mundo real, más cerca del contexto de las aplicaciones industriales o comerciales. Dónde cada hexágono corresponde a una zona de 500x500 m^2 reales. En la Figura 2.1, se pueden ver las dimensiones de una celda hexagonal, en base al área que se pretende modelar con ella.

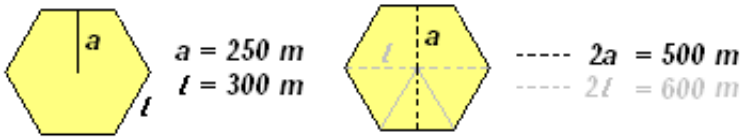

Figura 2.1: Dimensiones de una celda en base a las dimensiones reales en metros que se pretenden modelar. l es el lado y a es la apotema del hexágono.

Al modelado de un campo de batalla se le llamará mapa o escenario y será la representación de un problema concreto.

Del mismo modo, **la unidad** (CIA) a considerar en el problema, también ha sido trasladada al simulador. Teniendo en cuenta que ésta estará formada tanto por vehículos, como por soldados, pero que no podrá hacer uso de armas, solo moverse (pensemos en una unidad de salvamento, logística o de exploración). De forma que se considerará que dicha unidad ocupa exactamente una celda (según la doctrina militar, una CIA desplegada en formación de movimiento deberá ocupar una zona de 500x500 m^2). Además, se han asignado dos atributos a la misma:

- *puntos de recursos*: número que representa la cantidad de consumibles y los niveles de combustible de que dispone la unidad para moverse.

- *puntos de salud*: (o energía, si la denominamos según el término comúnmente aplicado en ciertos entornos, como los videojuegos) los cuales representan el estado de salud de los soldados de la unidad y el estatus de sus vehículos como un solo número.

Se considerará la posibilidad de que haya uno o varios **enemigos** en el campo de batalla, los cuales no tendrán propiedades asociadas, pero si que influirán en gran medida en la calidad de las soluciones encontradas, dado que de su situación dependerá la seguridad de los caminos que ellos vigilen. Además también afectarán a las zonas del mapa que puedan batir con sus armas. Dichos enemigos serán *Secciones Militares*, tres veces menores que la unidad que nos ocupa, pero que en la realidad ocupan el mismo espacio

desplegadas en formación defensiva que una CIA (500x500 m^2), por lo que también ocuparán exactamente una celda en el escenario presentado en el simulador.

Cada **celda** dentro del escenario/mapa tiene una serie de propiedades:

- *Tipo*: existen cuatro tipos de celdas: normal (tierra), bosque, agua y obstáculo (celda que la unidad no es capaz de atravesar).

- *Altura*: número entero entre -3 y 3 que representa el nivel de profundidad (negativo) o elevación (positivo) de la celda.

- *Letalidad*: Nivel de 0 a 100 asociado a la peligrosidad de dicha celda. Ese nivel se calcula para las celdas batidas por armas enemigas, como combinación de tres factores: la probabilidad de que el enemigo dispare, la probabilidad de que un disparo alcance a la celda y el daño que dicho disparo produciría a la unidad (potencia del arma).

Además, se podrá asignar un *subtipo* a cada celda, lo que la convertirá en la celda que ocupa la unidad (celda origen del camino a buscar), la celda de destino (celda final del camino a buscar) o la celda que ocupa un enemigo.

La Figura 2.2 muestra un ejemplo de campo de batalla. Éste incluye los elementos típicos: unidad enemiga, zona de impacto de armas, obstáculos, puntos de origen y destino y los tres tipos de celdas de terreno con diferentes alturas.

Para continuar con la definición del problema se han asignado dos **penalizaciones a cada celda** (costes relativos a cada uno de los objetivos a minimizar):

- *coste en recursos*: representa la dificultad que supone atravesar dicha celda. Depende de su tipo. Además se considerará un coste adicional cuando la unidad de mueva entre dos celdas con distintas alturas, siendo éste mayor cuando se pase de una de menor altura a una de mayor altura.

- *coste en salud/energía*: calculado como suma de dos cantidades independientes. Por un lado se tiene una penalización fija dependiente del tipo de celda, conocida en el entorno militar como *bajas de no combate*, la cual indica el consumo en salud provocado por dicha celda (los efectivos de la unidad van mermando debido a lesiones, heridas y al cansancio; los vehículos se desgastan y averían). Por otra parte se tiene

la *letalidad* asociada a la celda según su peligrosidad (impacto de armas enemigas).

Estas características están resumidas en la Tabla 2.1.

	Puntos Salud/Energía	*Puntos Recursos*
Compuestos por	salud global de los soldados, estado global de los vehículos	comida, combustible, medicinas, provisiones generales, moral
Consumidos por	bajas de no combate, letalidad	dificultad de atravesar celda, diferencia de altura

Tabla 2.1: Descripción de Salud/Energía y Recursos. Factores que constituyen los puntos de cada tipo y factores que consumen dichos puntos.

Dado que los puntos de la CIA se van consumiendo conforme recorre la ruta, entonces para esta modelización del problema, los objetivos iniciales que debían cumplir los caminos solución (maximizar rapidez y maximizar seguridad) pasarán a ser: **la minimización de costes en recursos y en salud** respectivamente.

Como aclaración, añadir que a la hora de considerar un camino como *rápido*, se equiparará a que se *consuman pocos recursos* en el transcurso del mismo, ya que eso indicará que se han atravesado menos celdas o celdas con menor dificultad (la penalización de recursos en las celdas indica dificultad para cruzarlas), y a velocidad constante, eso indicará también menos tiempo. Del mismo modo, al considerar un camino como *seguro*, se entenderá que atravesarlo *consume poca salud (energía)* de la unidad y que discurre por el mayor número de celdas ocultas a los enemigos posible para evitar la letalidad.

Hay que señalar un factor de extrema importancia, y es que los objetivos podrán tener una prioridad asignada, la cual estará prefijada antes de la resolución del problema y que depende de un parámetro al que se ha denominado λ. Dicha prioridad será complementaria, de modo que si se aumenta la de un objetivo, se disminuirá la del otro y viceversa, aunque es posible definir la misma para ambos. El objetivo de este parámetro es ofrecer cierta flexibilidad al usuario que pretenda resolver el problema, para que las soluciones se adapten a la situación que él ha planteado o está analizando, en la que es posible que deba primar la rapidez o la seguridad según su propio criterio (o siguiendo las reglas estipuladas por la doctrina militar).

Por último, en la modelización del problema se definieron una serie de **restricciones**, tanto para hacerlo afrontable dentro de un entorno de simulación, como para aumentar la fidelidad al modelo real:

- tanto la CIA, como cada unidad enemiga ocuparán exactamente una celda.
- una celda solo podrá ser de un tipo (y de un subtipo opcionalmente).
- podrá haber o no enemigos en el escenario.
- para definir completamente el problema es necesario indicar un punto origen y un punto destino, pero solo uno de cada tipo.
- la unidad puede atravesar una celda una única vez en su itinerario y no puede atravesar celdas ocupadas por enemigos (salvo que ese sea su objetivo), obstáculos, ni celdas cuyos costes en recursos o energía superen los disponibles de la unidad alcanzado ese punto.

Del mismo modo se han implementado una serie de *funciones y reglas de simulación* para hacer el modelo lo más realista posible:

- línea de visión: para determinar qué celdas ven los enemigos.
- capacidad de adquisición: límite hasta el que puede ver la unidad del problema y la unidad enemiga (puede ser distinta).
- obstáculos naturales: diferencias de altura que la unidad del problema no puede atravesar.

Además, el simulador permite la definición de problemas a partir de mapas reales, creando una capa de información subyacente, que es la que se trata para resolver el problema. Se puede ver un ejemplo de esto en la Figura B7, dentro del Apéndice B. El cual puede ser a su vez consultado para ampliar todos los conceptos comentados en esta sección.

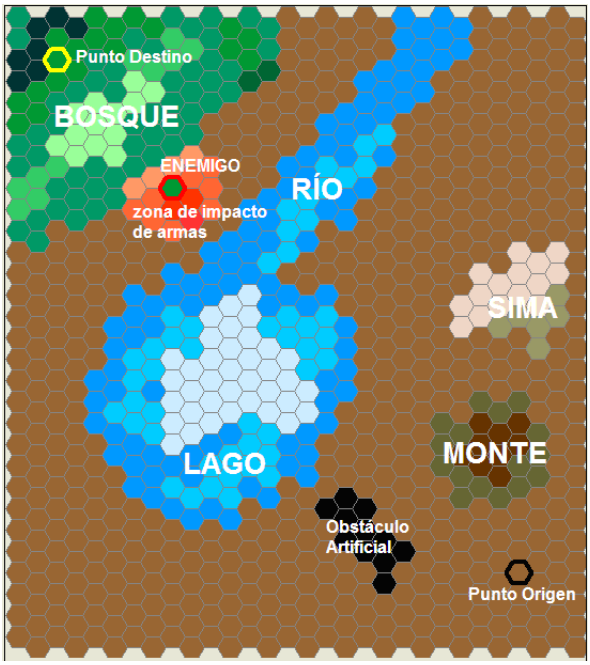

Figura 2.2: Mapa de ejemplo de 30x30 celdas que muestra los tipos de celdas y sus colores correspondientes (marrón-normal, verde-bosque, azul-agua, negro- obstáculo). Se han etiquetado los elementos modelados para identificarlos mejor. Los niveles de color indican la altura de las celdas (colores claros indican profundidad, los oscuros elevación). El punto origen es la celda con borde negro y el destino la de borde amarillo. La unidad enemiga aparece con borde rojo y las celdas alrededor de ella corresponden a la zona de impacto de sus armas. Los colores indican la letalidad de las mismas (cuanto más oscuros, más letales serán).

Capítulo 3

Optimización basada en Colonias de Hormigas

3.1. Introducción

Según se describe en varios estudios de Pierre-Paul Grassé [7, 8, 9], las hormigas son insectos sociales que viven en colonias y que basan su comportamiento en la colaboración para el beneficio de toda la colonia, en lugar del beneficio propio. Un aspecto muy interesante del comportamiento de las mismas, confirmado también por Denebourg et al. [10, 11], es su habilidad para encontrar los caminos más cortos entre su hormiguero y las fuentes de comida, lo cual resulta mucho más interesante sabiendo que muchas especies de hormigas son ciegas (o casi), por lo que se puede descartar la vista como mecanismo para la búsqueda de dichos caminos.

La explicación radica, considerando los estudios antes citados y las conclusiones alcanzadas en [12], en que mientras buscan la comida (una vez fuera del nido), las hormigas depositan una sustancia química denominada *feromona*, que todas pueden 'oler' y que se evapora con el tiempo. En principio, si no encuentran ningún rastro de feromona, las hormigas se mueven de manera prácticamente aleatoria, pero cuando encuentran un rastro de feromona depositada, tienden a seguirlo. La elección entre distintos caminos tiene lugar en las bifurcaciones, en las que las hormigas eligen el camino a seguir con una decisión probabilística muy dependiente de la cantidad de feromona, de modo que cuanto más fuerte es el rastro de feromona, mayor es la probabilidad de elegirlo. La comunicación entre las hormigas hecha a través del medio

en el que se mueven, es conocida como *estigmergia* [13].

Además, durante el movimiento de las hormigas siguiendo los rastros que detectan, se produce un proceso de autorrefuerzo ya que éstas depositan su propia feromona, lo que concluye en la formación de rastros señalados por una concentración de feromona elevada. Todo este mecanismo permite a las hormigas encontrar los caminos más cortos entre su hormiguero y la fuente del alimento, ya que dichos caminos serán visitados con mayor frecuencia por un mayor número de hormigas (al estar más cerca) y, por lo tanto, tendrán una concentración de feromona muy alta, puesto que la evaporación de la misma será menor que en los caminos más largos.

Como aclaración, decir que la evaporación de la feromona no es tan rápida como cabría pensar, ya que depende de muchas condiciones (tipo de sustancia según la especie de hormiga, tipo y estado del suelo, climatología), por lo que podría permanecer incluso varios días, pudiendo utilizarla las hormigas como una especie de 'memoria', para volver a buscar siguiendo los mismos caminos en otras ocasiones.

Por otra parte, dentro de las ciencias de la computación, existe una rama que se basa en la aplicación de mecanismos o comportamientos observados en la naturaleza a la creación de métodos o heurísticas para la resolución de problemas de computación complejos. Los algoritmos que surgen de dichos estudios son los llamados *algoritmos bioinspirados*.

La observación de los comportamientos de las hormigas en general y sus métodos de colaboración a la hora de buscar comida más concretamente, también han dado como fruto la aparición de una metaheurística relativamente reciente (presentada por Dorigo et al.[30] en 1991), la llamada *Optimización basada en Colonias de Hormigas* (OCH, en adelante y ACO, *Ant Colony Optimization*, en inglés) [14]. La cual está fuertemente inspirada en el comportamiento de dichos insectos a la hora de encontrar los mejores caminos entre el hormiguero y las fuentes de comida.

A continuación, se presentarán las características principales de los OCH, partiendo desde su inspiración en las hormigas naturales, pasando por su modo de funcionamiento y enumerando finalmente los modelos más importantes. Todo ello, siguiendo el esquema que se expone en el trabajo de Alonso et al. [60].

3.2. Hormigas como agentes artificiales de computación

Como se ha comentado en el apartado anterior, la metaheurística OCH está inspirada en el comportamiento que tienen ciertas especies de hormigas a la hora de establecer las mejores rutas entre el nido y las fuentes de comida. Estos algoritmos son ampliamente utilizados para la resolución de problemas de optimización complejos.

En ellos se consideran las hormigas como agentes computacionales simples, conocidos como 'hormigas artificiales', los cuales cooperan entre si comunicándose mediante rastros de feromona también artificiales (suele considerarse una matriz con los índices de la concentración en cada punto, modelados como números reales). Si bien el procesamiento que realiza cada hormiga/agente se hace de manera independiente al resto (pudiendo ser incluso paralelo). Normalmente se utiliza una colonia de estas hormigas, considerando un número de individuos dependiente de la dificultad del problema a resolver.

3.2.1. Hormigas naturales y hormigas artificiales

Dado que este modelo es una abstracción de ciertos esquemas de comportamiento dados en la naturaleza a la hora de buscar caminos mínimos, las colonias de hormigas naturales y artificiales compartirán una serie de características, como por ejemplo (ver [61]):

- Ambas utilizan una colonia de individuos que interaccionan y colaboran para solucionar una tarea dada.
- Todas tienen un mismo fin: la búsqueda del camino más corto (construcción iterativa de una solución de costo mínimo) desde un origen, el hormiguero (estado inicial), hasta un estado final, la comida (estado objetivo).
- Tanto las hormigas naturales como las artificiales hacen uso de una comunicación *estigmérgica*, es decir, modifican el medio en el que se mueven depositando feromona que las demás podrán detectar. En el caso de las segundas, los rastros de feromona artificiales son valores numéricos que se tienen en una memoria compartida y que se consideran únicamente de manera local.

- Ambas construyen sus soluciones paso a paso (iterativamente) aplicando una estrategia de transición local semi-estocástica (está ponderada por el nivel de concentración de feromona) para moverse entre estados adyacentes.

Aún así, se debe considerar que como metaheurística, la OCH incluye algunas características para potenciar los algoritmos construidos basados en ella que no tienen una correspondencia natural. Por ejemplo (según [61]):

- Las hormigas artificiales pueden hacer uso, en la política de transición que empleen, del conocimiento heurístico que se tiene a priori del problema a resolver, y no solo de los rastros locales de feromona.

- Estos agentes computacionales pueden considerar restricciones o condiciones en la búsqueda, para desechar de antemano ciertas soluciones.

- Tienen una memoria que almacena el camino seguido por la hormiga.

- La cantidad de feromona depositada por la hormiga artificial es proporcional a la calidad de la solución encontrada. Aunque en la naturaleza, ciertas especies de hormigas también depositan una mayor cantidad si la fuente de alimento encontrada es grande.

- El momento en el que se deposita la feromona es diferente en algunos casos, ya que, si bien hay algoritmos en los que se deposita en el momento en el que la hormiga artificial pasa a un nuevo estado, por lo general este aporte lo hacen una vez que han generado una solución completa.

- La evaporación de los rastros de feromona artificial es diferente a como se produce en la naturaleza, dado que en esta última, y como ya se comentó previamente, es posible que estos prevalezcan durante mucho tiempo (días e incluso semanas). En los algoritmos de OCH, se hace una evaporación (o varias) en cada iteración del algoritmo (cuyo impacto varía de unos a otros), ya que este mecanismo resulta fundamental para evitar el estancamiento en óptimos locales. De esa forma, las soluciones 'aprendidas' por las hormigas son 'olvidadas' poco a poco, a fin de redirigir la búsqueda hacia nuevas zonas del espacio de soluciones en iteraciones posteriores.

- Los algoritmos OCH pueden disponer de mecanismos adicionales para mejorar muchos de sus defectos, como por ejemplo la *optimización local* [16], que mejora la eficacia, o las *listas de candidatos* [16] que son conjuntos de los nodos vecinos más prometedores, las cuales ayudan a mejorar la eficiencia del algoritmo.

3.2.2. Problemas resolubles con OCH

Los tipos de problemas que pueden ser resueltos inicialmente utilizando OCH, son los conocidos como *problemas de camino mínimo* (PCM), un caso concreto de los problemas de búsqueda de camino óptimo (PCO) descritos en la sección 2.1.1. Estos problemas se caracterizan porque el objetivo a optimizar es la distancia recorrida entre dos puntos, la cual debe minimizarse. Aunque puede generalizarse a la minimización de cualquier función de coste que se quiera valorar al moverse entre dos puntos.

Los PCO pueden tener asociadas ciertas restricciones/condiciones, cuyo conjunto se notará Ω a la hora de considerarlo en un OCH (según Dorigo et al.[14]).

Por tanto, y como se comenta en dicha sección, este tipo de problemas pueden ser reformulados como búsquedas de rutas dentro de grafos finitos, conectados total o parcialmente, dirigidos o no, y con pesos asociados a los arcos (aristas).

En OCH se notará al conjunto de arcos del grafo A (nuevamente según se expone en [14]) y se denominará *vecindario* de un nodo al conjunto de los nodos conectados directamente con él (debe existir un arco entre cada uno de esos nodos y el que estemos considerando).

Por tanto, para resolver un problema mediante un OCH, éste debe convertirse en un grafo que cumpla esas propiedades.

3.2.3. La hormiga artificial

De forma que los algoritmos de OCH son esencialmente *algoritmos constructivos*, es decir, en cada iteración del algoritmo, cada hormiga de manera autónoma, construye una solución al problema recorriendo un grafo. Para ello, la hormiga va añadiendo en cada paso un nodo, elegido siguiendo un criterio en base a la información de que dispone. De este modo, cada arco del grafo, que representa los posibles pasos que la hormiga puede dar, tiene asociados dos tipos de información que guían el movimiento de la misma:

- **Información heurística**: es una medida de lo deseable que es dicho arco en base a la información previa que se tiene del problema (conocimiento heurístico), esto es, la preferencia de recorrer la arista a_{ij} (pasar del nodo i al nodo j). Las hormigas no modifican esta información, la cual se nota como $\eta(i, j)$

- **Información memorística** *(o información de los rastros de feromona)*: mide lo deseable que es el arco a_{ij} en base a lo aprendido por las hormigas que lo hayan seguido previamente. Éstas modifican dicha información simulando la actualización de feromona que se hace en la naturaleza. La modificación depende habitualmente de la calidad de la solución encontrada por cada una de las hormigas. Se nota como τ_{ij}.

De este modo, la *hormiga artificial* es un agente computacional simple que intenta construir soluciones posibles al problema explotando los rastros de feromona disponibles y la información heurística. Tiene las siguientes propiedades [14]:

- Busca soluciones para el problema a resolver, que sean válidas y tengan un coste mínimo.

- Tiene una lista o memoria L que almacena información sobre el camino seguido hasta el momento, esto es, L almacena la secuencia de nodos/arcos que ha generado (para comprobaciones, evaluaciones o reconstrucciones posteriores).

- Tiene un estado inicial $\delta_{inicial}$ (secuencia de inicio), y una o más condiciones t de parada.

- Comienza en el estado inicial y se mueve siguiendo arcos hasta nodos alcanzables (estados válidos), construyendo la solución asociada de manera incremental.

- Cuando llega a un estado δ_i (es decir, actualmente está localizada en el nodo i), puede moverse a cualquiera de los nodos de su vecindario posible N_i, definido como el conjunto de nodos que son alcanzables desde i siguiendo algún arco existente. Dichos nodos deberán cumplir con las condiciones/restricciones del problema para ser alcanzables.

- Tiene acceso a una estructura de datos en memoria compartida (τ) en la que se almacenan los valores de los rastros de feromona para cada uno de los arcos del grafo problema. Este es el único mecanismo de comunicación entre las hormigas.

- El movimiento se lleva a cabo aplicando una *regla de transición*, que es una función dependiente de los rastros de feromona visibles localmente, de los valores heurísticos que conoce la hormiga y de las restricciones del problema.

- Si durante el procedimiento de construcción una hormiga se mueve desde el nodo i hasta el j, ésta podrá actualizar el rastro de feromona τ_{ij} asociado al arco a_{ij}. Este proceso se llama *actualización en línea de los rastros de feromona paso a paso* o actualización local de feromona.

- El procedimiento de construcción acaba cuando se satisface alguna condición de parada, normalmente cuando se alcanza un estado (o nodo) objetivo.

- Una vez que la hormiga ha construido la solución, puede reconstruir el camino recorrido (según su memoria L) y actualizar los rastros de feromona de los arcos visitados utilizando un proceso llamado *actualización en línea a posteriori* o actualización global de feromona.

3.3. Estructura general de un algoritmo OCH

El modo de operación básico de un algoritmo de OCH es como sigue:

Las m hormigas (artificiales) de la colonia se mueven, concurrentemente y de manera asíncrona, a través de los nodos adyacentes del grafo (representan los estados del problema). Este movimiento se realiza siguiendo una regla de transición que está basada (se guía) en la información local, tanto heurística, como memorística (rastros de feromona) disponible en los nodos. De este modo, las hormigas construyen incrementalmente soluciones. Opcionalmente, las hormigas pueden depositar feromona cada vez que crucen un arco (conexión) mientras que construyen la solución (*actualización en línea paso a paso de los rastros de feromona*). Una vez que cada hormiga ha generado una solución, ésta se evalúa y puede depositar una cantidad de feromona que es función de la calidad de su solución (*actualización en línea de los rastros de feromona*).

La actualización en línea se suele llamar *actualización local de feromona*. Esta información guiará la búsqueda de las otras hormigas de la colonia en el futuro.

Además, el modo de operación genérico de un algoritmo de OCH incluye dos procedimientos adicionales, la evaporación de los rastros de feromona y las acciones del demonio. La evaporación de feromona la lleva a cabo el entorno y se usa como un mecanismo que evita el estancamiento en la búsqueda y permite que las hormigas exploren nuevas regiones del espacio. Las acciones del demonio son acciones opcionales (no tienen una correspondencia natural) para implementar tareas desde una perspectiva global que no pueden llevar a cabo las hormigas por la perspectiva local que ofrecen. Por ejemplo, observar la calidad de todas las soluciones generadas y depositar una nueva cantidad de feromona adicional sólo en las transiciones/componentes asociadas a algunas soluciones, o aplicar un procedimiento de búsqueda local a las soluciones generadas por las hormigas antes de actualizar los rastros de feromona. El demonio puede reemplazar la actualización en línea a posteriori de feromona y el proceso pasa a llamarse *actualización fuera de línea de rastros de feromona (o actualización global de feromona)*.

La estructura de un algoritmo de OCH genérico es como sigue [14]:

Algoritmo 1 OCH()

```
Inicializar_parametros()
while criterio_de_terminacion_no_satisfecho do
  for cada hormiga k do
    Construir_Solucion(k)
  end for
  Evaporacion_de_Feromona()
  Acciones_del_demonio() {opcional}
end while
```

Como se puede ver en el pseudocódigo que describe el cuerpo principal del Algoritmo 1, el primer paso consiste en la inicialización de los valores de los parámetros que considerará el algoritmo, como son: el rastro inicial de feromona, τ_0, asociado a cada transición, que es un valor positivo pequeño y normalmente el mismo para todas las conexiones (aristas); el número de hormigas en la colonia, m, o los pesos que definen la proporción en la que afectarán la información heurística y memorística en la regla de transición probabilística.

Algoritmo 2 Construir_Solucion(id_hormiga)

```
inicializar_hormiga(id_hormiga)
estado_actual = estado_inicial
L = guardar(estado_inicial)
while estado_actual ≠ estado_objetivo do
  estado_siguiente= Regla_Transicion(estado_actual)
  mover_al_siguiente_estado(estado_siguiente)
  if actualizacion_feromona_en_linea_paso_a_paso then
    depositar_feromona_en_el_arco_visitado()
  end if
  L = guardar(estado_siguiente)
  estado_actual = estado_siguiente
end while
if actualizacion_feromona_en_linea_a_posteriori then
  for cada arco visitado do
    depositar_feromona_en_el_arco_visitado()
  end for
end if
```

Algoritmo 3 estado_sig Regla_Transicion(estado_act)

```
P = calcular_probabilidades_de_transicion(estado_act, A, L, Ω)
estado_sig = aplicar_politica_decision(P, Ω)
```

El bucle principal de la metaheurística OCH como se puede ver es bastante simple y consta de tres grandes pasos: la generación y puesta en funcionamiento de las hormigas artificiales (que buscan una solución válida cada una), la evaporación de feromona, y las acciones del demonio (que son opcionales). Como se puede observar, se usa una implementación secuencial, aunque debido a la filosofía del algoritmo y a la 'independencia' entre las hormigas, el paralelismo podría ser explotado de manera sencilla y eficiente.

El procedimiento `Construir_Solucion` (Algoritmo 2) es el que se encarga de que cada hormiga busque una solución, en él se hace nuevamente una inicialización, esta vez de los datos de la hormiga, como su memoria, el estado inicial del que partirá (puede ser fijo, diferente para cada hormiga, aleatorio, etc) u otros datos dependientes del problema. Posteriormente, ésta se sitúa en el estado inicial y entra en un bucle en el que va pasando de estado en estado, actualizando la feromona en línea paso a paso (si hay que hacerlo)

hasta llegar al objetivo, momento en el que se hace la actualización en línea a posteriori de feromona (si hay que hacerla). Para elegir el siguiente estado al que pasar, se usa una regla de transición. La solución será la serie de estados que está almacenada en la lista L de cada hormiga.

La función `Regla_Transicion` (Algoritmo 3) es la encargada de decidir el siguiente estado al que se debe pasar desde el que está situada la hormiga, para ello se calcula la probabilidad de pasar a cada estado posible en base a la lista de conexiones (aristas) A, la lista de estados visitados L y las restricciones del problema Ω, y a continuación, se aplica una política de decisión en base a dichas probabilidades y, nuevamente, las restricciones del problema.

Como se ha comentado anteriormente, varios de estos componentes son o bien opcionales (acciones del demonio), o bien dependientes estrictamente del algoritmo de OCH especifico (cuándo y cómo se deposita la feromona). Generalmente, la actualización en línea paso a paso de los rastros de feromona y la actualización en línea a posteriori de los rastros de feromona no suelen estar presentes a la vez ni faltar ambas al mismo tiempo (si las dos faltan, el demonio deberá actualizar los rastros de feromona).

3.4. Modelos principales

A continuación se comentarán los principales modelos de algoritmos OCH, comenzando por los originales introducidos por Dorigo et al. en [14, 5] y presentando después algunas propuestas posteriores.

3.4.1. El Sistema de Hormigas

El SH [15] (*Ant System*, en inglés), desarrollado por Dorigo, Maniezzo y Colorni en 1991, fue el primer algoritmo de OCH. Su estructura principal es la comentada en el apartado anterior, teniendo como característica que la actualización de feromona se realiza una vez que todas las hormigas han completado sus soluciones, y que se lleva a cabo en dos pasos: en primer lugar, todos los rastros de feromona se reducen en un factor constante (evaporación) y a continuación, cada hormiga de la colonia deposita una cantidad de feromona en función de la calidad de su solución (aporte).

Inicialmente, el SH no usaba ninguna acción del demonio, pero es relativamente fácil, por ejemplo, añadir un procedimiento de búsqueda local para

refinar las soluciones generadas por las hormigas.

Para construir las soluciones en el SH, como ya se ha comentado, se usa la llamada *regla de transición* en cada paso de construcción. De modo que una hormiga k, situada en un nodo i, escoge el siguiente nodo al que moverse j, con una probabilidad que se calcula como:

$$P_{ij}^k = \begin{cases} \dfrac{(\tau_{ij})^\alpha \cdot (\eta_{ij})^\beta}{\sum_{u \in N_i^k} (\tau_{iu})^\alpha \cdot (\eta_{iu})^\beta} & si\ j \in N_i^k \\ \\ 0 & en\ otro\ caso \end{cases} \tag{3.1}$$

donde N_i^k es el vecindario alcanzable por la hormiga k cuando se encuentra en el nodo i, y $\alpha, \beta \in \mathrm{R}$ son dos parámetros que ponderan la importancia relativa de los rastros de feromona y la información heurística respectivamente. Cada hormiga k almacena la secuencia que ha seguido hasta el momento en su memoria L^k y, tal como se explicó antes, ésta se utiliza para determinar N_i^k en cada paso de construcción.

Respecto a los parámetros α y β, su función es la que sigue: si α=0, aquellos nodos con una preferencia heurística mejor tienen una mayor probabilidad de ser escogidos, haciendo al algoritmo muy similar a un algoritmo voraz probabilístico clásico (con múltiples puntos de partida en caso de que las hormigas estén situadas en nodos distintos al comienzo de cada iteración). Sin embargo, si β=0, sólo se tienen en cuenta los rastros de feromona para guiar el proceso constructivo, lo que puede causar un rápido estancamiento, esto es, una situación en la que los rastros de feromona asociados a una solución son ligeramente superiores al resto, provocando por tanto que las hormigas siempre construyan las mismas soluciones, normalmente óptimos locales. Por tanto, es preciso establecer una adecuada proporción entre la información heurística y la información de los rastros de feromona.

Como se ha comentado anteriormente, la *actualización de la feromona* se realiza una vez que todas las hormigas han acabado de construir sus soluciones. En primer lugar se realiza una *evaporación* de los rastros de feromona en todos los arcos, reduciéndolos en un factor constante según la expresión:

$$\tau_{ij} = (1 - \rho) \cdot \tau_{ij} \qquad \forall\ a_{ij} \tag{3.2}$$

donde $\rho \in (0,1]$ es la tasa de evaporación (un parámetro configurable al inicio del algoritmo).

A continuación, cada hormiga k recorre nuevamente el camino que ha seguido (el cual está almacenado en su memoria local L^k) y deposita una cantidad de feromona en cada conexión a_{rs} por la que ha viajado, que será el *aporte*:

$$\tau_{rs} = \tau_{rs} + \Delta\tau_{rs}^k \qquad \forall\ a_{rs}\ \in\ S^k \tag{3.3}$$

siendo S^k la solución encontrada por la hormiga k. El aporte, $\Delta\tau_{rs}^k$, depende de la calidad de dicha solución, lo que se nota como:

$$\Delta\tau_{rs}^k = f(C(S^k)) \tag{3.4}$$

Los creadores del SH propusieron posteriormente una versión extendida del algoritmo, que normalmente mejoraba los resultados obtenidos y que denominaron *SH elitista* [15]. En dicho algoritmo, una vez que las hormigas han depositado feromona en las conexiones asociadas a sus respectivas soluciones, el demonio realiza una deposición adicional de feromona en las aristas que pertenecen a la mejor solución encontrada hasta el momento en el proceso de búsqueda (esta solución se denominará la mejor global de aquí en adelante). La cantidad de feromona depositada, que depende de la calidad de la mejor solución global, se incrementa en un factor e, que se corresponde con el número de hormigas elitistas que se consideran, tal como sigue:

$$\tau_{rs} = \tau_{rs} + e \cdot f(C(S_{mejor\_global})) \qquad \forall\ a_{rs}\ \in\ S_{mejor\_global} \tag{3.5}$$

3.4.2. El Sistema de Colonia de Hormigas

El SCH [16] (*Ant Colony System*, en inglés) fue uno de los primeros sucesores del SH, de hecho fue propuesto por los mismos autores. Introduce tres modificaciones importantes con respecto al algoritmo anterior.

En primer lugar, el SCH usa una regla de transición distinta, denominada *regla proporcional pseudo-aleatoria*: dada una hormiga k, situada en el nodo i, el siguiente nodo j se elige de manera aleatoria mediante la siguiente distribución de probabilidad:

`Si` $(q \leq q_0)$

$$j = \arg\max_{j \in N_i^k} \left\{ \sum_{u \in N_i^k} (\tau_{iu})^\alpha \cdot (\eta_{iu})^\beta \right\} \tag{3.6}$$

`Si no`

$$P_{ij}^k = \begin{cases} \dfrac{(\tau_{ij})^\alpha \cdot (\eta_{ij})^\beta}{\sum_{u \in N_i^k} (\tau_{iu})^\alpha \cdot (\eta_{iu})^\beta} & si\ j \in N_i^k \\ \\ 0 & en\ otro\ caso \end{cases} \tag{3.7}$$

siendo $q_0 \in [0,1]$ un parámetro llamado factor de proporción del SCH y q un valor aleatorio en [0,1]. El resto de los parámetros son los mismos que los utilizados en la Ecuación 3.1.

Como puede observarse, la regla tiene una doble intención: cuando $q \leq q_0$, se explota el conocimiento disponible, eligiendo la mejor opción con respecto a la información heurística y los rastros de feromona. Sin embargo, si $q > q_0$ se aplica una exploración controlada, tal como se hacia en el SH. En resumen, la regla establece un compromiso entre la exploración de nuevas conexiones y la explotación de la información disponible en ese momento. El parámetro q_0 fija la importancia que tendrá un factor sobre otro (la exploración respecto de la explotación).

La segunda diferencia es que únicamente el demonio (y no las hormigas individualmente) actualiza la feromona de forma global, es decir, se realiza una actualización de feromona fuera de línea de los rastros. Para llevarla a cabo, el SCH sólo considera una hormiga concreta, la que generó la mejor solución global, S_{mejor_global}.

Adicionalmente, el demonio puede aplicar un algoritmo de búsqueda local para mejorar las soluciones de las hormigas antes de actualizar los rastros de feromona.

Esta actualización de la feromona se hace evaporando primero los rastros en todas las conexiones utilizadas por la mejor hormiga global tal como sigue:

$$\tau_{rs} = (1 - \rho) \cdot \tau_{rs} \qquad \forall\ a_{rs}\ \in\ S_{mejor\_global} \tag{3.8}$$

A continuación el demonio realiza el aporte según la fórmula:

$$\tau_{rs} = \tau_{rs} + \rho \cdot f(C(S_{mejor\_global})) \qquad \forall\ a_{rs}\ \in\ S_{mejor\_global} \tag{3.9}$$

La tercera modificación que introduce el SCH respecto al SH es que las hormigas aplican una actualización en línea paso a paso de los rastros de feromona que favorece la generación de soluciones distintas a las ya encontradas.

Cada vez que una hormiga viaja por una arista a_{rs}, aplica la regla:

$$\tau_{rs} = (1 - \varphi) \cdot \tau_{rs} + \varphi \cdot \tau_0 \tag{3.10}$$

donde $\varphi \in (0,1]$ es un segundo parámetro de evaporación de feromona. Como puede verse, la regla de actualización en línea paso a paso incluye tanto la evaporación de feromona como el aporte de la misma. Dado que la cantidad de feromona depositada es muy pequeña (de hecho, τ_0 es el valor del rastro de feromona inicial), la aplicación de esta regla hace que los rastros de feromona entre las conexiones recorridas por las hormigas disminuyan, de modo que esto añade un factor de exploración adicional al SCH ya que las conexiones atravesadas por un gran número de hormigas son cada vez menos atractivas para el resto de las que las recorren en la iteración actual, lo que ayuda claramente a que no todas las hormigas sigan el mismo camino.

3.4.3. El Sistema de Hormigas Max-Min

El SHMM [62, 63] (*Max-Min Ant System*, en inglés), desarrollado por Stützle y Hoos en 1996, es una de las extensiones del SH que mejor rendimiento muestran. Extiende el SH en tres aspectos que se comentan a continuación.

El primero es que se aplica una actualización de los rastros de feromona fuera de línea, de manera similar a como se hace en el SCH. De modo que después de que todas las hormigas hayan construido su solución, cada rastro de feromona sufre una evaporación (según la Ecuación 3.2). A continuación, se deposita feromona siguiendo la siguiente fórmula:

$$\tau_{rs} = \tau_{rs} + f(C(S_{mejor})) \qquad \forall\, a_{rs} \;\in\; S_{mejor} \tag{3.11}$$

La *mejor hormiga* a la que se permite añadir feromona puede ser la que tiene la solución *mejor de la iteración* o la solución *mejor global*. Lo que se hace es elegir primero a las mejores de la iteración y, conforme avance el algoritmo, ir eligiendo a las mejores globales.

Además, en el SHMM las soluciones que ofrecen las hormigas suelen ser mejoradas usando optimizadores locales (búsqueda local) antes de la actualización de feromona.

En segundo lugar, los valores posibles para los rastros de feromona están limitados al rango $[\tau_{min}, \tau_{max}]$. Por lo tanto, la probabilidad de un estancamiento del algoritmo disminuye al darle a cada conexión existente una probabilidad, aunque pueda ser bastante pequeña, de ser escogida. En la práctica,

$\tau_{max} = 1/(\rho \cdot C(S^*))$, donde S^* es la solución óptima (se suele usar la mejor global). Para τ_{min}, normalmente sólo es necesario escoger su valor de tal manera que sea un factor constante menor que τ_{max}.

Para poder incrementar la exploración de nuevas soluciones, el SHMM utiliza en ocasiones reinicializaciones de los rastros de feromona (se hace cuando se estima que el algoritmo no avanza como debiera).

En tercer lugar, en vez de inicializar los rastros de feromona a una cantidad pequeña, el SHMM los inicializa a una estimación del máximo permitido para un rastro (la estimación puede obtenerse generando una solución S' con una heurística voraz y reemplazando S^* por S' en la ecuación de τ_{max}). Esto lleva a una componente adicional de diversificación en el algoritmo, ya que al comienzo las diferencias relativas entre los rastros de feromona no serán muy acusadas, lo que no ocurre cuando los rastros de feromona se inicializan a un valor muy pequeño.

3.4.4. El Sistema de Hormigas con Ordenación

El SHO [64] (*Rank-Based Ant System*, en inglés) es otra extensión del SH propuesta por Bullnheimer, Hartl y Strauss en 1997. Incorpora la idea de ordenar las hormigas para realizar la actualización de feromona, que el demonio realiza nuevamente, fuera de línea, tal como se comenta a continuación.

En primer lugar, las m hormigas se ordenan de mejor a peor según la calidad de sus soluciones: $(S'_1, \ldots, S'_m)$, siendo S'_1 la mejor solución construida en la iteración actual.

Tras esto, el demonio deposita feromona en las conexiones por las que han pasado las $(\sigma - 1)$ mejores hormigas (hormigas elitistas). La cantidad de feromona depositada depende directamente del orden de la hormiga y de la calidad de su solución.

Finalmente, las conexiones por las que ha pasado la mejor hormiga global reciben una cantidad adicional de feromona que depende únicamente de la calidad de dicha solución. Este aporte de feromona se considera el más importante y, de hecho, recibe un peso de σ.

Esta metodología de operación tiene efecto al utilizar la siguiente regla de actualización de feromona, la cual se aplica a cada arista una vez que todos los rastros de feromona han sido evaporados (usando nuevamente la Ecuación 3.2):

$$\tau_{rs} = \tau_{rs} + \Delta\tau_{rs}^{mg} + \Delta\tau_{rs}^{orden} \tag{3.12}$$

donde:

$$\Delta\tau_{rs}^{mg} = \begin{cases} f(C(S_{mejor\_global})) & si\ a_{rs} \in S_{mejor\_global} \\ \\ 0 & en\ otro\ caso \end{cases} \tag{3.13}$$

$$\Delta\tau_{rs}^{orden} = \begin{cases} \sum_{\mu=1}^{\sigma-1}(\sigma-\mu)\cdot f(C(S'_{\mu})) & si\ a_{rs} \in S'_{\mu} \\ \\ \\ 0 & en\ otro\ caso \end{cases} \tag{3.14}$$

3.5. Aplicaciones de los algoritmos OCH

Los algoritmos de OCH se han aplicado a un gran número de problemas de optimización combinatoria diferentes, como se indica en [49]. Las aplicaciones actuales de la OCH se distribuyen dentro de dos clases fundamentales: los *problemas de optimización combinatoria NP-duros* y los *problemas dinámicos de caminos mínimos*.

La primera clase de problemas la componen un grupo de problemas para los que las técnicas clásicas ofrecen a menudo un comportamiento algo pobre.

La segunda clase de aplicaciones se caracterizan porque la instancia del problema que hay que resolver cambia durante la ejecución del algoritmo. Estos cambios pueden afectar a la topología del problema, como por ejemplo la disponibilidad de los enlaces o, si la topología del problema es fija, características como los costes de los arcos pueden variar con el tiempo. En este caso, el algoritmo tiene que adaptarse a la dinámica del problema. Un ejemplo de esta última clase sería el enrutamiento en redes de comunicaciones.

Una característica común a casi todas las aplicaciones exitosas de la OCH es la combinación de las hormigas con algoritmos de búsqueda local que refinan las soluciones ofrecidas por estas.

En lugar de enumerar exhaustivamente las diversas aplicaciones de esta metaheurística, se va a describir brevemente la evolución histórica de los problemas que se han resuelto aplicándola.

El primer problema que se planteó resolver con un algoritmo de OCH fue el TSP [24], ya que éste es una instancia bien conocida de un problema NP-duro, que además incluye de manera inmediata un problema de camino

mínimo, haciendo por tanto que su adaptación al comportamiento real de las hormigas para resolverlo fuera una tarea directa. Desde la primera aplicación del SH en la memoria de la tesis de Dorigo en 1991, se convirtió en un problema estándar para realizar pruebas en otros modelos posteriores que ofrecían un mejor rendimiento que el SH.

Cronológicamente, las dos aplicaciones siguientes fueron el problema de la asignación cuadrática (QAP) [65] y el problema de la secuenciación de tareas (*jobshop scheduling*, JSP) [66] en 1994.

Entre las aplicaciones posteriores se encuentran las primeras de enrutamiento en redes, comenzando en 1996 con el trabajo de Schoonderwoerd [67] y otros, y el trabajo sobre AntNet de Di Caro y Dorigo [68].

Algunas aplicaciones de principios de 1997 (aunque algunas de ellas aparecieron publicadas más tarde) incluyen problemas clásicos de enrutamiento de vehículos [69], de ordenación secuencial [70], de secuenciación (*flow shop scheduling*, FSS) [71] y de coloreo de grafos [72].

Desde entonces, muchos autores distintos han usado la metaheurística OCH para solucionar un gran número de problemas de optimización combinatoria como la supersecuencia común más corta, la asignación generalizada, la cobertura de conjuntos, varios problemas de la mochila y de satisfacción de restricciones, entre otros. Aparte de las aplicaciones anteriores, los algoritmos de OCH han sido utilizados recientemente para aprendizaje automático (*machine learning*), concretamente para el diseño de algoritmos de aprendizaje de estructuras de representación del conocimiento como las clásicas reglas lógicas [73], reglas difusas [74] y redes bayesianas [75], demostrando resultados bastante prometedores.

Actualmente, la OCH es capaz de obtener los mejores resultados para varios de los problemas a los que ha sido aplicada, tales como QAP, ordenación secuencial, enrutamiento de vehículos, secuenciación, y enrutamiento de paquetes en redes, entre otros. Los resultados computacionales para otros muchos problemas son muchas veces muy buenos y cercanos a los mejores.

Para concluir, comentar que la metaheurística OCH está siendo aplicada a nuevos problemas reales con resultados prometedores, por ejemplo, al diseño de circuitos lógicos combinatorios [76].

Capítulo 4

Problemas y algoritmos multiobjetivo

En en mundo real existen gran cantidad de problemas que se caracterizan por la existencia de múltiples *medidas de adecuación*, las cuales deberían ser optimizadas, o al menos, ser satisfechas simultáneamente. Es decir, problemas cuya solución tiene varias componentes que muchas veces están en conflicto, por lo que la mejora de una de ellas conlleva un empeoramiento en algunas (o todas) las demás. Un ejemplo sería un sistema de refrigeración de un congelador que pretendiese mantener la temperatura lo más baja posible pero con el menor consumo de energía que se pudiera. Estos objetivos estarían en conflicto, pues para conseguir un descenso de temperatura sería necesario consumir más energía.

En este tipo de problemas es difícil saber si una solución es mejor que otra, pues es posible que una optimice unos objetivos en detrimento de otros y otra solución distinta lo haga al revés, por tanto, la elección de la solución adecuada quedaría en muchos casos sujeta al criterio del diseñador del problema. En el ejemplo anterior no se podría decidir a priori si es mejor una solución que obtenga una temperatura mínima de -20º C con 200W u otra que obtenga -15º C con 100W y sería el diseñador del sistema de refrigeración el que decidiese qué solución le interesaba más.

4.1. Definición de un problema multiobjetivo

Un PMO (MOP, *Multi-Objective Problem* en inglés) se puede definir matemáticamente como [6]:

> *"un vector de variables de decisión que satisface un conjunto de restricciones y optimiza un vector función cuyos elementos representan las funciones objetivo. Estas funciones forman una descripción matemática de los criterios de rendimiento de un sistema determinado, que normalmente están en conflicto entre ellos. Así, el término 'optimizar' significa encontrar una solución que de a cada función objetivo un valor aceptable para el diseñador"*

Formalmente, un PMO se define como la minimización o maximización de una función:

$$F(x) = (f_1(x), ..., f_k(x)) \tag{4.1}$$

sujeta a una serie de restricciones $g_i(x) \ \ \forall i = 1...m$ donde $x \in \Omega$ y siendo x un vector n-dimensional $x = (x_1, ..., x_n) \in X$.

Habitualmente los objetivos estarán en conflicto entre sí, de forma que la mejora de uno de ellos sólo se podrá dar empeorando otro. De hecho, encontrar el óptimo global en un PMO es un problema NP-completo [77]. En la mayoría de los casos no existe un óptimo global (que todas las variables satisfagan las restricciones, y todas las funciones objetivo alcancen simultáneamente su óptimo global).

Un concepto muy importante dentro de este tipo de problemas es la **dominancia**. Una solución a (vector solución, en realidad) domina a una solución b (considerando la minimización de las funciones objetivo/funciones de coste), y se nota $a \prec b$ si:

$$\forall i \in 1, 2, ..., k \mid f_i(a) \leq f_i(b) \ \ \wedge \ \ \exists j \in 1, 2, ...k \mid f_j(a) < f_j(b) \tag{4.2}$$

Es decir, la solución a es mejor o igual que la b en todos los objetivos y, al menos, mejor en alguno de ellos.

En el campo de los PMO, el concepto de óptimo tiene un significado diferente al dado en los problemas con un solo objetivo. Se suele utilizar el

concepto de optimalidad propuesto por Edgeworth [78] y generalizado por Pareto [79]. Así,un vector (solución) $x^* \in F$ es *Pareto-óptimo* si:

$$\begin{array}{l} \nexists x \in F \ \ tal\ que \ \ f_i(x) \leq f_i(x^*) \ \ \forall i = 1, ..., k \\ y\ \exists j \ \ tal\ que \ \ f_j(x) < f_j(x^*) \end{array} \tag{4.3}$$

Es decir, no existe otro vector (solución) que lo domine, por lo que a estos vectores se les conoce como soluciones *no dominadas*.

En este tipo de problemas, por tanto, no hay una única solución óptima, sino un conjunto de soluciones (normalmente grande e incluso infinito), de las que posteriormente se podrían seleccionar varias si se desea. Este conjunto de soluciones es el conocido como *Conjunto de Pareto* y la representación gráfica de las funciones objetivo de dichos vectores es el llamado *Frente de Pareto* (FP en adelante). Un ejemplo de éste puede verse en la Figura 4.1.

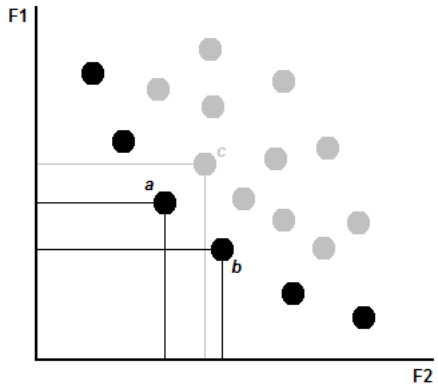

Figura 4.1: Ejemplo de Frente de Pareto en un problema con dos funciones objetivo (F1 y F2). Los puntos negros representan las soluciones del Frente de Pareto (a y b entre ellas) de un problema con 2 objetivos (F1 y F2). Las soluciones dominadas aparecen en color gris (por ejemplo c).

Como se puede ver en la figura, las soluciones del FP no son dominadas por ninguna otra, mientras que las demás soluciones (en gris) son dominadas por una o varias de las de este frente. Si nos fijamos en las soluciones a, b y c (las dos primeras dentro del frente), se pueden comprobar ambas condiciones:

- a es mejor (tiene un valor menor, ya que se está minimizando) que b

en el objetivo F2, pero tiene un valor peor en F1 por lo que ninguna domina a la otra.

- c es mejor que b en F2, pero peor en F1, entonces no es dominado por b, pero c tiene peores valores que a en ambos objetivos, por lo que a si que domina a c.

Otro dato a tener en cuenta es que normalmente, para afrontar un problema de este tipo se hace una 'transformación' de las funciones objetivo de forma que todas tiendan maximizarse o minimizarse. Para pasar de maximizar a minimizar una función (o viceversa) basta con hacer la transformación $max(f_i(x)) = min(-f_i(x))$.

Normalmente, a la hora de resolver estos problemas, se busca encontrar el mayor número de soluciones del frente de Pareto que sea posible, pues todas serán válidas y 'optimas' (siempre teniendo en cuenta que ninguna es mejor que otra dentro del frente).

4.2. Algoritmos multiobjetivo

Se trata de algoritmos desarrollados para resolver problemas del tipo de los expuestos en el apartado anterior. Se suelen agrupar en dos grandes grupos según se basen en el concepto de dominancia de Pareto o no. El primer grupo no se basa en dicha dominancia, son técnicas eficientes, pero algo obsoletas y limitadas a un uso sobre tres objetivos máximo para evitar malos comportamientos. Las más conocidas son:

- *Ordenación Lexicográfica*: antes de buscar soluciones se hace una ordenación de los objetivos, de forma que se empieza a buscar solución para el objetivo más importante en primer lugar. Una vez se haya encontrado su 'óptimo' (o la solución que se estime como buena), se empieza la búsqueda para el segundo objetivo, considerando como valor para el primero la solución obtenida previamente. Este proceso se repite para todos los objetivos siguiendo el orden establecido. Es una técnica eficiente por su simplicidad, pero su gran problema es que depende mucho de la ordenación que se haga [80].

- *Combinación Lineal de Pesos*: se hace una ponderación de los objetivos con pesos que indiquen la importancia relativa y se hace una sumatoria

de todos los objetivos, con lo que se convierte un problema multiobjetivo en monoobjetivo. Nuevamente, se trata de una técnica simple y eficiente, pero con problemas como por ejemplo el hecho de que los objetivos utilicen medidas en unidades distintas, con lo que hay que normalizarlos todos y elegir bien el conjunto de pesos. Además el uso de técnicas agregativas está muy mal visto en el ambiente de los PMO [81].

- ϵ-*Constraint*: en este caso se optimiza la función objetivo preferida, y se consideran las funciones adicionales como restricciones acotadas por ciertos niveles permisibles a los que se denomina ϵ_i. Por tanto, se efectúa una optimización mono-objetivo sujeta a restricciones adicionales de la función objetivo elegida. Posteriormente se cambian los niveles ϵ_i a fin de generar el conjunto completo de Pareto. Es un método simple, pero costoso computacionalmente debido a los grandes niveles de variabilidad que se requieren [82].

Las técnicas basadas en dominancia de Pareto, consideran que las soluciones no dominadas deberían tener un mayor valor en la jerarquía de ordenación de soluciones, de forma que en un algoritmo genético, por ejemplo, estas soluciones serían los mejores individuos de la población. En este caso, es muy recomendable usar alguna técnica para mantener la diversidad, ya que se busca obtener el mayor número de soluciones posibles del frente de Pareto. En los apartados siguientes se presentarán diversos algoritmos, tanto evolutivos, como de OCH, que usan métodos basados en la dominancia de Pareto.

4.2.1. Algoritmos evolutivos multiobjetivo

En un principio los AEs fueron el método más eficiente para atacar este tipo de problemas, ya que al estar basados en una población de soluciones candidatas, son capaces de generar un conjunto de soluciones Pareto-óptimas en cada ejecución del algoritmo. A continuación se describen algunos de los AEMO más extendidos [83] que utilizan diversas técnicas basadas en Pareto:

- **MOGA** (Multi-Objective Genetic Algorithm). Implementado por Fonseca y Fleming [84]. Emplea un esquema de ranking basado en el frente

de Pareto (el rango de un individuo viene dado por el número de individuos de la población que lo dominan). Incorpora la técnica de *fitness sharing*.

- **MOMGA** (Multi-Objective Messy Genetic Algorithm). Implementado por Van Veldhuizen [85]. Desarrollado para explorar la relación entre los bloques constructivos de las soluciones multiobjetivo (para reducir el número de bloques constructivos) y su uso en la búsqueda del MOEA. Incorpora *fitness sharing* y la selección de torneo propuesta por Horn et al. [86].

- **MOMGA-II**. Propuesto por Zydallis y Lamont [87]. Extiende el MOMGA usando la técnica *Probabilistically Complete Initialization* (PCI) y una fase de filtrado de bloques constructivos.

- **NPGA** (Niched-Pareto Genetic Algorithm). Implementado por Horn et al. [86]. Utiliza selección de torneo basada en el concepto de Pareto-optimalidad, y *fitness sharing*.

- **NPGA-II**. Propuesto por Erickson et al. [88], mejora NPGA añadiendo Pareto ranking.

- **NSGA** (Nondominated Sorting Genetic Algorithm). Implementado por Srinivas y Deb [89]. Emplea un esquema de ranking basado en el frente de Pareto, e incorpora la técnica de *fitness sharing*. Está basado en varias capas de clasificación de los individuos.

- **NSGA-II**. Propuesto por Deb et al. [90]. Mejora NSGA haciéndolo más eficiente computacionalmente. Incorpora elitismo y *crowding* para mantener la diversidad.

- **PAES** (Pareto Archived Evolution Strategy) y **PAES-II**. Implementado por Knowles y Corne [91, 92]. Usa un solo individuo-padre que genera un solo descendiente (estrategia evolutiva). Utiliza un fichero externo y un procedimiento para mantener la diversidad. PAES-II utiliza regiones objetivo para seleccionar individuos.

- **SPEA** (Strength Pareto Evolutionary Algorithm). Implementado por Zitzler y Thiele [93]. Desarrollado para explorar el uso de los individuos en el frente de Pareto en la asignación generacional del fitness. Emplea

un esquema de ranking basado en el frente de Pareto, e incorpora la técnica de *fitness sharing*. Usa una población secundaria en el proceso de asignación del fitness.

- **SPEA-2**. Propuesto por Zitzler et al. [94]. Utiliza asignación del fitness de grano fino, estimación de la densidad y truncación del archivo externo.

- **SFGA** (Single Front Genetic Algorithm). Desarrollado por De Toro et al. [95]. Usa un esquema de selección elitista conjuntamente con un método de aclarado que facilita una implementación eficiente.

Existen otras implementaciones, en las que se proponen diferentes técnicas de selección, asignación del fitness o de *clustering*, aunque todos se basan en la misma idea general.

4.2.2. Algoritmos de Optimización basada en Colonias de Hormigas multiobjetivo

En la bibliografía se pueden encontrar diferentes algoritmos de OCH diseñados para afrontar PMOs [49]. Entre los más extendidos destacan los siguientes:

- **MOAQ** (Multiobjective Ant-Q). Propuesto por Mariano y Morales en [96] y aplicado al diseño de redes de distribución de agua. Se basa en el Ant-Q [31], un SCH que trabaja con agentes que van pasando por estados en los que realizan acciones, obteniendo una recompensa en base al estado y a la acción. Además se cuenta con una heurística que valora cómo de bueno sería realizar una determinada acción en cada estado. La variante multiobjetivo cuenta con una familia de agentes (hormigas) para cada objetivo. A la hora de buscar una solución cada familia tiene en cuenta las soluciones encontradas por las demás y la recompensa depende de dichas soluciones. Se utiliza un conjunto externo para guardar las soluciones no dominadas (elitismo).

- **BicriterionAnt**. Diseñado por Iredi et al. [97] para resolver el problema del enrutamiento de vehículos bi-criterio. Se trata de un SH que usa dos matrices de feromonas y dos funciones heurísticas, una por cada objetivo, con una sola colonia de hormigas. En cada iteración, cada

hormiga construye una solución utilizando una regla de transición que combina la heurística y la feromona de ambos objetivos, ponderados (el par feromona-heurística de cada objetivo) con un parámetro λ que varía para cada hormiga (empieza en 0 para la primera y llega a valer 1 para la última) y que hace que cada una de ellas pondere más la información de un objetivo que de otro, para especializarse en una zona del frente de Pareto. Solo las hormigas con solución no dominada actualizarán las dos matrices de feromona. Al igual que en el algoritmo anterior, se usa un conjunto externo para implementar el elitismo guardando las soluciones no dominadas.

- **MultiColony para MO**. Propuesto por Iredi en el mismo artículo [97], y para el mismo problema bi-criterio (se puede generalizar a n objetivos). En él se tiene una colonia de hormigas por cada objetivo, cada una con su matriz de feromonas. De forma que cada hormiga solo actualiza una matriz, bien la de su propia colonia (con lo que las colonias se especializan en zonas del espacio de soluciones) o bien la que corresponda a la zona del frente de Pareto asociada a la solución que haya encontrado (previamente se divide el frente en regiones y se asocia cada región a una colonia, con lo que se obliga a cada colonia a buscar en una zona concreta).

- **P-ACO** (Pareto Ant Colony Optimization). Descrito por Doerner en [98] para resolver el problema de la selección multiobjetivo de carpeta. Se trata de un SCH en el que se usan nuevamente k matrices de feromonas (una por objetivo), pero la actualización global solo la hacen las dos mejores hormigas de la iteración en cada objetivo. En cada iteración, cada hormiga calcula un vector de pesos y los usa para combinar la información heurística y la feromona de todas las matrices (de todos los objetivos). Mientras se construye el camino se van evaporando los rastros de feromona en cada matriz (en cada objetivo) y al final de la iteración, las dos mejores hormigas en cada objetivo actualizan los rastros con más aporte en los arcos que estén en ambas soluciones, un poco menos si solo están en la mejor, menos a los que solo estén en la segunda mejor y 0 al resto.

- **MACS** (Multiple Ant Colony System). Propuesto por Barán et al. [43]. Se trata nuevamente de un SCH que usa una única matriz de feromonas y varias funciones heurísticas (una por objetivo). La regla de

transición combina la feromona con la información heurística de cada objetivo, ponderadas éstas últimas con un parámetro λ, al igual que en el BicriterionAnt. Durante la construcción de las soluciones, se va evaporando feromona y aportando en base a un τ_0 que es calculado como 1 partido por el producto de los fitness medios de cada objetivo para las soluciones que hay en el frente de Pareto. Este valor se va adaptando, pues cada vez que se incluye una nueva solución no dominada en el frente, cambia. Si el nuevo valor obtenido es menor que el anterior (y si se está minimizando), se reinician los rastros de feromona con ese nuevo valor, en caso contrario, las soluciones no dominadas hacen un aporte global que depende nuevamente de ese valor medio del fitness para cada objetivo de dichas soluciones.

Capítulo 5

Algoritmos estudiados y desarrollados para resolver el problema

En este capítulo se describen en detalle cada uno de los algoritmos diseñados para la resolución del problema que nos ocupa. Estos métodos propuestos se presentan como el núcleo de esta tesis y pretenden ser una aportación importante dentro del campo de los algoritmos de OCH para la resolución de problemas de camino óptimo multicriterio, constituyendo incluso un buen método para la resolución de problemas multiobjetivo en general.

Los algoritmos presentados son distintas propuestas formuladas para afrontar el mismo problema, pero postulados como algoritmos que consideran un número de objetivos distinto, constituyendo tres algoritmos específicos:

- *CHAC*: algoritmo propuesto para la resolución del problema tal y como se ha formulado en el Capítulo 2. Por tanto se trata de un algoritmo de OCH, concretamente un SCH, diseñado para trabajar con dos criterios (considera dos matrices de feromona, dos funciones heurísticas y dos funciones de evaluación). Además, puede utilizar dos reglas de transición diferentes, una agregativa y otra que sigue un enfoque multiobjetivo 'puro', por lo que se podría considerar como dos algoritmos diferentes (al utilizar una regla u otra).

- *mono-CHAC*: algoritmo diseñado para la resolución del problema, pero considerando un único objetivo (obtenido a partir de una combinación

de los dos criterios originales). Se puede ver como una reformulación del algoritmo anterior. Todos los elementos del algoritmo CHAC se rediseñan para su funcionamiento como un algoritmo de OCH más parecido al clásico SCH (pensado inicialmente para resolver problemas con una sola función objetivo).

- *CHAC-4*: algoritmo desarrollado para la resolución del problema bicriterio, pero considerando una división en subobjetivos de cada uno de los principales, por lo que este método está diseñado para trabajar con cuatro objetivos. Todos los elementos que CHAC consideraba por partida doble, CHAC-4 los utilizará por cuadruplicado. Aunque, al igual que aquel, puede aplicar dos reglas de transición diferentes, siendo nuevamente una agregativa y otra con criterios separados, por lo que de nuevo se podría considerar como dos algoritmos diferentes.

Además, en el capítulo también se presenta *CHAC-N*, una generalización de estos métodos planteada para la resolución de problemas multiobjetivo con cualquier número de criterios.

5.1. Algoritmo CHAC

CHAC significa Compañía de Hormigas Acorazadas, nombre con el que se pretende relacionar los algoritmos OCH con el ámbito militar y con la unidad que nos ocupa. Se trata de un SCH adaptado para tratar con dos objetivos, es decir, un algoritmo OCHMO (Optimización basada en Colonias de Hormigas Multiobjetivo, MOACO en inglés).

A modo de recordatorio (ver Sección 2.2.1), decir que el problema a resolver es la búsqueda del camino óptimo de una unidad militar (una CIA) dentro de un campo de batalla atendiendo a dos criterios, encontrar un camino que sea rápido y que sea seguro (el nivel de rapidez y seguridad lo fija un parámetro que determina el usuario). Dicha unidad tiene asignados unos recursos que serían combustible o consumibles y un nivel de salud (o energía) que serían sus efectivos. Ambos se van consumiendo en el camino en base a las condiciones del mismo (penalizaciones en las celdas, diferencias de alturas, impacto de armas). También se debe recordar que se habla de un camino *rápido*, cuando en él *se consumen pocos recursos*, y al considerar un camino como *seguro*, se entiende que atravesarlo *consume poca salud*

(energía) de la unidad y que discurre por el mayor número posible de celdas ocultas a los enemigos.

A la hora de resolver un problema mediante un algoritmo OCH, como se comentó en el Capítulo 3, es necesario llevar a cabo una transformación de dicho problema para que quede representado como un grafo cuyas aristas tengan asignados pesos. En este caso, dado que el problema está representado por una rejilla de celdas, que puede tener diferente topología, cada celda del mapa se considerará un nodo del grafo y tendrá ocho vecinos (si las celdas son cuadradas) o seis vecinos (si las celdas son hexagonales). Se consideran vecinos al grupo de celdas que rodean a una dada y que están conectadas con ella mediante arcos (aristas). Como es lógico, las celdas de los límites tendrán menos vecinos (véase la Figura 5.1).

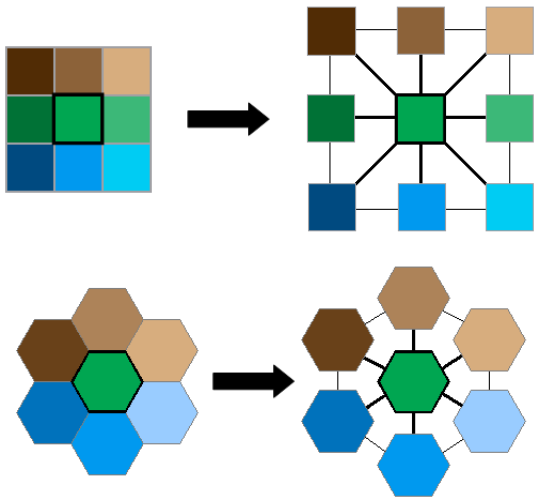

Figura 5.1: Conversión de vecindario de celdas a grafo. Cada celda será un nodo y estará conectada con sus vecinos (celdas que la rodean). Ejemplos en una rejilla cuadrada y otra hexagonal respectivamente.

En una primera aproximación, se trabajó con un grid *cuadrado*, es decir, se consideraba una rejilla donde cada celda tenía ocho vecinos. Sobre dicho entorno se implementó el algoritmo CHAC. Posteriormente, se consideró una modelización del entorno más realista, mediante el uso de un grid *hexagonal*, donde cada celda tenía seis vecinos. Este es el modelo que se seguirá en este trabajo. Además, se adaptó el algoritmo inicial al nuevo grid, constituyéndose

el que se bautizó como *hCHAC*. Aún así y dado que son en esencia el mismo, en este trabajo se hará referencia al algoritmo únicamente como CHAC.

Para poder tratar dos objetivos, habrá dos pesos asociados a cada arista que indicarán el coste en recursos y en energía que se consumirían al atravesarla. El coste en recursos depende del tipo de la celda destino (que tiene asociado un consumo de recursos) y de la diferencia de alturas, mientras que el coste en energía vendrá dado por el tipo de la celda destino (tiene asociado un consumo de energía) y la *letalidad* de la misma.

A continuación se describirá el algoritmo propuesto, poniendo especial atención a las funciones heurísticas, reglas de transición, actualización de feromona y funciones de evaluación (elementos principales en un algoritmo de OCH).

5.1.1. Descripción detallada del algoritmo

El algoritmo que implementa CHAC es constructivo, lo que significa que en cada iteración, cada hormiga construye una solución completa (como se comentó en el Capítulo 3), si es posible, ya que hay algunas restricciones, como que un camino no debe pasar dos veces por el mismo nodo, que no se pueden salvar ciertos obstáculos (naturales o artificiales) y que no se puede pasar a un nodo cuyo coste agote los recursos o salud que le restan a la unidad (se van consumiendo conforme se construye cada camino para comprobar que es factible). Esta solución se construye moviéndose por el grafo.

Para guiar este movimiento, el algoritmo usa dos tipos de información: los *rastros de feromona* y la *información heurística* que son combinadas.

CHAC considera *dos matrices de feromonas*, una por objetivo y *dos funciones heurísticas* (que también son matrices calculadas al principio), siguiendo el planteamiento del algoritmo BicriterionAnt diseñado por Iredi et al. en [97]. En este caso se utiliza también *una única colonia de hormigas*, pero el algoritmo implementa un SCH en lugar del SH básico que usan ellos, a fin de tener un mayor control en el equilibrio entre exploración y explotación por medio del parámetro estándar q_0.

Se han implementado dos reglas de transición de estados diferentes (lo que significa dos algoritmos distintos), una similar a la propuesta por Iredi y la otra basada en dominancia de vecinos (considerando la información de los objetivos por separado). Las fórmulas de actualización local y global de feromona son parecidas a las del algoritmo MACS-VRPTW (Multi-SCH para resolver el problema MO del enrutamiento de vehículos con ventana de

tiempo), propuesto por Barán et al. en [43], solo que con algunos cambios a fin de usar dos matrices de feromonas.

Los objetivos son: la minimización de los recursos consumidos en el camino (maximización de rapidez), al que se ha llamado **r**, y la minimización de la salud consumida en el camino (maximización de seguridad), denominado **s**.

5.1.1.1. Funciones heurísticas

Estas funciones tratan de guiar la búsqueda entre el punto origen y el punto destino, considerando los factores más importantes para cada objetivo. Para el arco *(i,j)* serían:

$$\eta_r(i,j) = \frac{\omega_r^r}{R(i,j)} + \frac{\omega_r^d}{d(j,T)} + (\omega_r^o \cdot O(j)) \tag{5.1}$$

$$\eta_s(i,j) = \frac{\omega_s^s}{S(i,j)} + \frac{\omega_s^d}{d(j,T)} + (\omega_s^o \cdot O(j)) \tag{5.2}$$

En la Ecuación 5.1, R es el coste en recursos cuando se pasa del nodo i al nodo j y d es la distancia Euclídea entre dos nodos (T es el nodo de destino del problema).

O[1] es un número (entre 0 y 1) que valora la ocultación de una celda y que tomará su valor en función de dos posibilidades:

- *si hay enemigos conocidos* (definidos en el mapa): será 1 si la celda está oculta a todos los enemigos. Este valor decrece exponencialmente cuando es vista (cuantos más enemigos la vean, menor es).

- *si no hay enemigos conocidos*: se calcula la visibilidad de dicha celda desde cada una de las celdas dentro de un área determinada, normalmente considerando como radio la distancia que indica la capacidad de adquisición de la CIA. De modo que se considerará como valor la media de celdas a las que es oculta, siendo 1 si es oculta a todas.

ω_r^r , ω_r^d y ω_r^o son pesos que asignan la importancia relativa a cada uno de los miembros de la fórmula. En este caso, el miembro más importante es la distancia al punto de destino, ya que se está buscando el camino más rápido (el que menos pasos de); en segundo lugar, es importante el consumo

[1]Está basada en la función que determina la Línea de Visión, descrita en el Apéndice B5

de recursos y en menor medida la ocultación de la celda. En la Ecuación 5.1, S es el coste de salud de pasar al nodo j, d y O son los mismos que en la ecuación anterior. ω_s^s , ω_s^d y ω_s^o son nuevamente pesos para asignar la importancia relativa a los miembros de la fórmula, pero en este caso los más importantes son el coste de salud y la ocultación de la celda (ambos deben ser considerados en un camino seguro) y mucho menos la distancia al destino.

Es importante notar que el cometido de las funciones heurísticas es valorar lo deseable que es pasar de un nodo i a un nodo j, tomando un valor mayor cuanto más deseable sea este movimiento. De esta forma, los términos de estas funciones toman valores inversos a los que se pretenden minimizar, para así tomar valores mayores cuanto menores sean los costes (R y S) o la distancia al destino.

Del mismo modo, se considera una función de ocultación (en lugar de visibilidad) en la función heurística porque a mayor ocultación, más deseable sería moverse al nodo j que valora la heurística. Si se hubiese considerado la visibilidad, una gran visibilidad conllevaría un mayor valor para la heurística, lo cual no sería deseable en este problema.

5.1.1.2. Reglas de transición de estados

Estas reglas son las que deciden el siguiente nodo j al que pasar cuando se está construyendo un camino (ver Sección 3.3) y se está situado en el nodo i. En este caso, se han diseñado dos reglas diferentes para tratar con el problema, ambas son reglas multiobjetivo, por lo que el uso de una u otra dará como resultado dos algoritmos diferentes.

Regla de Transición Combinada (RTC). Se trata de la regla proporcional pseudo-aleatoria que se usa normalmente en los SCH, pero adaptada para trabajar con dos objetivos combinando la información heurística y la feromona de ambos. La regla es:

Si $(q \leq q_0)$

$$j = \arg\max_{j \in N_i} \left\{ \tau_r(i,j)^{\alpha \cdot \lambda} \cdot \tau_s(i,j)^{\alpha \cdot (1-\lambda)} \cdot \eta_r(i,j)^{\beta \cdot \lambda} \cdot \eta_s(i,j)^{\beta \cdot (1-\lambda)} \right\} \qquad (5.3)$$

Sino

$$P(i,j) = \begin{cases} \dfrac{\tau_r(i,j)^{\alpha\cdot\lambda}\cdot\tau_s(i,j)^{\alpha\cdot(1-\lambda)}\cdot\eta_r(i,j)^{\beta\cdot\lambda}\cdot\eta_s(i,j)^{\beta\cdot(1-\lambda)}}{\sum_{u\in N_i}\tau_r(i,u)^{\alpha\cdot\lambda}\cdot\tau_s(i,u)^{\alpha\cdot(1-\lambda)}\cdot\eta_r(i,u)^{\beta\cdot\lambda}\cdot\eta_s(i,u)^{\beta\cdot(1-\lambda)}} & si\, j \in N_i \\ 0 & en\, otro\, caso \end{cases} \quad (5.4)$$

Dónde $q_0 \in [0,1]$ es el parámetro estándar del SCH y q es un valor aleatorio en [0,1]. τ_r y τ_s son las matrices de feromona de los objetivos y η_s y η_s son las funciones heurísticas de dichos objetivos (Ecuaciones 5.1 y 5.2). Todas estas matrices tienen un valor para cada arco (i,j). α y β son los parámetros de ponderación usuales y N_i es el conjunto de vecinos alcanzables (cumplen las restricciones) para el nodo i.

$\lambda \in (0,1)$ es un parámetro cuyo valor da el usuario y que fija la importancia relativa de los objetivos en la búsqueda (esta aplicación ha sido creada para un usuario militar que decidirá qué objetivo tiene más prioridad y cuánta). De modo que, por ejemplo, si el usuario decide buscar el camino más rápido, λ tendrá un valor cercano a 1, y si lo que quiere es el camino más seguro, deberá tomar un valor cercano a 0. Este valor se mantiene durante toda la ejecución del algoritmo y es el mismo para todas las hormigas, a diferencia de otras implementaciones bi-criterio en las que dicho parámetro tiene un valor 0 para la primera hormiga y va tomando valores cada vez mayores hasta llegar a tomar 1 para la última hormiga (según la fórmula $\lambda_k = \frac{k-1}{m-1}$, para cada hormiga $k \in [1,m]$). Con este enfoque, siempre se busca en una zona determinada del espacio de búsqueda (la que nos interesa).

El funcionamiento de la regla es sencillo: cuando una hormiga, que está construyendo una solución, se encuentra situada en el nodo i, si $q \leq q_0$, el mejor vecino j (el que mayor valor tenga para la expresión de la Ecuación 5.3) es seleccionado como siguiente. En otro caso, el algoritmo decide a qué nodo pasar mediante una ruleta de selección de probabilidades, considerando $P(i,j)$ como probabilidad para cada nodo alcanzable j (Ecuación 5.4).

Regla de Transición basada en Dominancia (RTD). Esta regla se basa en el concepto de dominancia visto en el Apartado 4.1 (Ecuación 4.2), pero considerando como funciones objetivo dos funciones de coste. Dichas

funciones consideran arcos que son los que tienen asignados la información heurística y memorística y la combinan:

$$C_r(i,j) = \tau_r(i,j)^{\alpha} \cdot \eta_r(i,j)^{\beta} \tag{5.5}$$

$$C_s(i,j) = \tau_s(i,j)^{\alpha} \cdot \eta_s(i,j)^{\beta} \tag{5.6}$$

Si se tiende a la maximización en lugar de a la minimización (como se comentó en la Ecuación 4.2), se tendría para dos vectores a y b, con dos valores, uno por objetivo:

$$\begin{array}{l} a \succ b \;\; sii: \\ \forall x \in [r,s] \mid C_x(a) \geq C_x(b) \;\; \wedge \;\; \exists y \in [r,s] \mid C_y(a) > C_y(b) \end{array} \tag{5.7}$$

De modo que se ha definido una función que hace uso de este último concepto, considerando los arcos del grafo (como vectores de dos valores a comparar) y operando teniendo en cuenta las funciones de coste definidas en las Ecuaciones anteriores (5.5 y 5.6).

$$D(i,j,u) = \begin{cases} 1 & si\ (i,j) \succ (i,u) \\ \\ 0 & en\ otro\ caso \end{cases} \tag{5.8}$$

Por último, la regla de transición basada en dominancia será:

Si $(q \leq q_0)$

$$j = \arg\max_{j \in N_i} \left\{ \sum_{u \in N_i} D(i,j,u) \quad \forall j \neq u \right\} \tag{5.9}$$

Sino

$$P(i,j) = \begin{cases} \dfrac{\left(\displaystyle\sum_{u \in N_i} D(i,j,u) \right) + 1}{\displaystyle\sum_{k \in N_i} \left(\left(\sum_{u \in N_i} D(i,k,u) \right) + 1 \right)} & si\ j \in N_i \wedge j \neq u \wedge k \neq u \\ \\ 0 & en\ otro\ caso \end{cases} \tag{5.10}$$

Dónde N_i es nuevamente el vecindario de i (nodos alcanzables desde él). Esta regla elige el siguiente nodo j en el camino (cuando la hormiga está situada en el nodo i) considerando el número de vecinos dominados por cada uno de los arcos que se pueden formar desde i con sus vecinos. De modo que si $q \leq q_0$, se elige como siguiente, el nodo cuyo arco con i domine a más (arcos) vecinos de i (Ecuación 5.9). En caso contrario se usa una ruleta de selección considerando el número de (arcos) vecinos de i que domina cada arco como probabilidad asociada (Ecuación 5.10). En esta última ecuación se añade un 1 en el numerador y en el denominador para evitar una probabilidad 0 si un determinado arco no domina a ningún otro o una indeterminación si ningún arco domina a ningún otro.

5.1.1.3. Funciones de evaluación

Son las funciones que asignan un coste global a cada solución encontrada por una hormiga. Consideran, no sólo el coste en recursos o salud de pasar a cada nodo (celda) en el camino solución, sino también la visibilidad de dicha celda, ya que ésta es de suma importancia en todo tipo de caminos, pues el discurrir por celdas visibles al enemigo podría significar grandes perjuicios para la unidad (emboscadas, ataques en campo abierto, etc), incluso en los caminos en los que deba primar la rapidez. Eso si, la importancia de este término no será la misma en ambos casos, claro está.

Nuevamente, se tienen dos funciones de evaluación (una por objetivo):

$$F_r(C_{sol}) = \sum_{n \in C_{sol}} [R(n-1,n) + \omega^o_{F,r} \cdot (1 - O(n))] \tag{5.11}$$

$$F_s(C_{sol}) = \sum_{n \in C_{sol}} [S(n-1,n) + \omega^o_{F,s} \cdot (1 - O(n))] \tag{5.12}$$

Dónde C_{sol} es el camino solución que se va a evaluar y $\omega^o_{F,r}$, $\omega^o_{F,s}$ son pesos relativos a la importancia de la visibilidad/ocultación de las celdas del camino. En la Ecuación 5.11 la importancia será poca (no es muy importante mantenerse oculto en un camino rápido), por contra en la Ecuación 5.12 será muy importante, como es lógico en un camino seguro. Los demás términos son iguales que en las Ecuaciones 5.1 y 5.2.

5.1.1.4. Actualización de feromona

Dado que CHAC es un SCH, hay dos niveles de actualización de feromona, local y global. En cada uno de ellos se actualizan dos matrices de feromona.

Actualización local de feromona. Se lleva a cabo cada vez que un nuevo nodo j es añadido al camino (posible solución) que una hormiga está construyendo; por tanto, ésta se aplica al nuevo arco incluido en el camino, (i, j).

El objetivo de esta actualización es evaporar los rastros de feromona en el nodo (y hacer un pequeñísimo aporte), para disuadir a las demás hormigas de que sigan el mismo camino que la que realiza la actualización. De este modo, se intenta que las demás busquen en otras zonas del espacio de búsqueda, es decir, busquen caminos alternativos, lo que significa un aumento de la exploración.

Ésta se lleva a cabo con las expresiones:

$$\tau_r(i,j)^t = (1-\rho)\cdot\tau_r(i,j)^{t-1} + \rho\cdot\tau_{0,r} \tag{5.13}$$

$$\tau_s(i,j)^t = (1-\rho)\cdot\tau_s(i,j)^{t-1} + \rho\cdot\tau_{0,s} \tag{5.14}$$

Se ha señalado con t el nuevo valor para la feromona y con $t-1$ el anterior. Además, $\rho \in [0,1]$ es el factor de evaporación común y $\tau_{0,r}$, $\tau_{0,s}$ son las cantidades iniciales de feromona en cada arco para cada objetivo, las cuales constituyen el aporte mínimo que se pretendía hacer sobre la feromona de los arcos ya visitados por la hormiga actual. Dichas cantidades iniciales se calculan respectivamente como:

$$\tau_{0,r} = \frac{1}{(num\_cels\cdot MAX_R)} \tag{5.15}$$

$$\tau_{0,s} = \frac{1}{(num\_cels\cdot MAX_S)} \tag{5.16}$$

Siendo num_cels el número de celdas del escenario a resolver, MAX_R el número máximo de recursos que puede requerir atravesar una celda y MAX_S el máximo coste de salud (energía) que puede requerir pasar por una celda (en el peor de los casos).

Actualización global de feromona. Esta actualización se hará considerando únicamente las soluciones dentro del Frente de Pareto (soluciones no dominadas), es decir, se aplicará a los arcos que formen parte de dichos caminos. Por tanto, ésta se llevará a cabo una vez que todas las hormigas hayan acabado de construir sus soluciones y se hayan valorado las posibles dominancias entre ellas.

El objetivo que persigue esta actualización es el de potenciar los buenos caminos para que las hormigas de las siguientes iteraciones consideren sus arcos como buenas opciones en la búsqueda de soluciones óptimas, lo que se traduce en un aumento de la explotación en el algoritmo.

Las formulas utilizadas en esta actualización son:

$$\tau_r(i,j)^t = (1-\rho)\cdot\tau_r(i,j)^{t-1} + \rho/F_r \tag{5.17}$$

$$\tau_s(i,j)^t = (1-\rho)\cdot\tau_s(i,j)^{t-1} + \rho/F_s \tag{5.18}$$

Nuevamente se ha señalado la nueva feromona con t y la anterior con $t-1$. ρ es el mismo parámetro que se usó para la actualización local, F_r y F_s son los costes asociados a la solución, calculados usando las Ecuaciones 5.11 y 5.12.

Como se puede observar, el aporte de feromona será directamente proporcional a la calidad de la solución a la que corresponda el arco a actualizar, es decir, inversamente proporcional a la función que se pretende minimizar.

5.1.2. Pseudocódigo

El pseudocódigo para el algoritmo CHAC se puede ver en los Algoritmos 4 y 5.

5.2. Algoritmo mono-CHAC

La versión monoobjetivo del algoritmo implementado inicialmente (y comentado en el Apartado 5.1) se ideó con el fin de probar si el problema que nos ocupa en este estudio podría ser resuelto satisfactoriamente como una función agregativa de los objetivos deseados. Es decir, dicho problema podría ser planteado como un problema de búsqueda de camino óptimo con un único objetivo, creando una nueva función a minimizar que combinase los criterios de rapidez y seguridad ponderándolos con sendos pesos en función de su importancia, esto es:

$$F(C_{sol}) = \omega_r \cdot F_r(C_{sol}) \;\; + \;\; \omega_s \cdot F_s(C_{sol}) \tag{5.19}$$

Algoritmo 4 CHAC()

```
Inicializar_feromona(τ0,r, τ0,s) {Ecuaciones 5.15 y 5.16}
for número de iteraciones do
  for cada hormiga k do
    {Ecuaciones 5.1, 5.2, [5.3, 5.4] o [5.9, 5.10], 5.13, 5.14}
    camino_completo=Construir_Solucion(k)
    if camino_completo=TRUE then
      Evaluar(k.solucion) {Ecuaciones 5.11 y 5.12}
      if dominado(Frente_Pareto, k.solucion)=FALSE then
        insertar(Frente_Pareto, k.solucion)
        eliminar_soluciones_dominadas(Frente_Pareto, k.solucion)
      end if
    end if
  end for
  Actualizacion_Global_Feromona(Frente_Pareto) {Ecuaciones 5.17 y 5.18}
end for
```

Siendo ω_r y ω_s los pesos que ponderarían a las funciones a minimizar para obtener el camino más rápido y el más seguro respectivamente.

Para ello, se ha planteado nuevamente el problema considerando que solo hay un objetivo a minimizar (combinación de los dos iniciales) y se ha considerado su resolución mediante un SCH adaptado al mismo, al que se ha llamado *mono-CHAC*.

Tanto la transformación del problema en grafo (requisito de los algoritmos OCH), como los diferentes costes, restricciones y propiedades descritas al principio de la Sección 5.1 se mantienen iguales.

A continuación se detallará el algoritmo propuesto, poniendo nuevamente especial atención a los elementos principales de cualquier OCH, como son la función heurística, regla de transición, actualización de feromona y función de evaluación.

5.2.1. Descripción detallada del algoritmo

mono-CHAC es nuevamente un algoritmo constructivo (dado que es un OCH), por lo que en cada iteración, cada hormiga construirá una solución completa si le es posible (a veces no podrá por las restricciones existentes). En dicha construcción se guiará según los valores *heurísticos* y *memorísticos*, como ya se comentó en el Capítulo 3.

Algoritmo 5 Construir_Solucion(a)

```
{inicializaciones de niveles de recursos y salud}
a.recursos=UNIDAD.recursos
a.salud=UNIDAD.salud
{primer nodo del camino}
nodo_actual = NODO_INICIAL
a.solucion.añadir(nodo_actual)
fin=FALSE
while fin=FALSE do
  {considera los recursos y energía restante, y los obstáculos}
  N_i=Nodos_Alcanzables(nodo_actual, a)
  {hay algún nodo alcanzable}
  if N_i ≠ [ ] then
    {se elige el siguiente nodo usando una regla de transición}
    {RTC: Ecuaciones 5.3 y 5.4 o RTD: Ecuaciones 5.9 y 5.10}
    nodo_siguiente=Seleccionar_Nodo_Siguiente(nodo_actual, N_i)
    {se añade el siguiente nodo al camino solución}
    a.solucion.añadir(nodo_siguiente)
    {se calculan los consumos}
    a.recursos=a.recursos - Coste_Recursos(nodo_actual, nodo_siguiente)
    a.salud=a.salud - Coste_Salud(nodo_actual, nodo_siguiente)
    {Ecuaciones 5.13 y 5.14}
    Actualizacion_Local_Feromona(nodo_actual, nodo_siguiente)
    {la hormiga 'se mueve' al siguiente nodo}
    nodo_actual=nodo_siguiente
    {se comprueba si se ha llegado al final}
    if nodo_siguiente=NODO_DESTINO then
      fin=TRUE
      camino_completo=TRUE
    end if
  else
    fin=TRUE
    camino_completo=FALSE
  end if
end while
return(camino_completo)
```

En este caso, únicamente se hará uso de una matriz de feromonas y de una única función heurística, ya que el problema solo tiene un objetivo a minimizar. En este caso también se implementará un SCH, para volver a aprovechar las posibilidades del parámetro de equilibrado entre exploración y explotación, q_0.

Al ser un algoritmo monoobjetivo, únicamente habrá una solución considerada como la mejor hasta el momento. Dicha solución es la que se utilizará para realizar la actualización a nivel global de feromona. De la misma forma, el algoritmo únicamente obtendrá una solución final, que será la mejor de todas las soluciones encontradas por las hormigas durante toda la ejecución del mismo.

5.2.1.1. Función heurística

Al ser un algoritmo diseñado para resolver un problema con un único objetivo, hará uso de una sola función heurística, la cual debe guiar a las hormigas basándose en la información que se tiene del problema a priori para minimizar la función objetivo. Dicha función será, al igual que la función objetivo, el resultado de la agregación de los diferentes componentes de las funciones heurísticas presentadas para dos objetivos (Ecuaciones 5.1 y 5.2):

$$\eta(i,j) = \frac{\omega_r}{R(i,j)} + \frac{\omega_s}{S(i,j)} + \frac{\omega_d}{d(j,T)} + (\omega_o \cdot O(j)) \tag{5.20}$$

Dónde R y S son, al igual que en las Ecuaciones 5.1 y 5.2, el coste en recursos y salud cuando se pasa del nodo i al j, d es la distancia Euclídea entre dos nodos (T es el nodo destino) y O es la función que valora la ocultación (comentada anteriormente). ω_r, ω_s, ω_d y ω_o son pesos que asignan la importancia relativa de cada uno de los miembros de la fórmula. Los valores para los dos primeros son los mismos que en el caso del algoritmo CHAC bi-criterio y los valores de los dos últimos se han calculado como la media de los valores que tomaban los pesos respectivos en dichas ecuaciones. Es decir:

$$\begin{aligned} \omega_r &= \omega_r^r \qquad \omega_s = \omega_s^s \\ \omega_d &= \tfrac{\omega_r^d + \omega_s^d}{2} \quad \omega_o = \tfrac{\omega_r^o + \omega_s^o}{2} \end{aligned} \tag{5.21}$$

De forma que todos los miembros mantendrán la misma importancia.

Como se puede ver, la función resultante consta de cuatro miembros, pues se ha obtenido al combinar las funciones heurísticas de la implementación bi-objetivo (CHAC). Dichas funciones heurísticas contaban, además de con los

términos puramente referidos a los consumos de recursos (R) y de salud (S), con miembros referidos a la distancia al punto destino (d, que debía reducirse en ambos casos) y la ocultación de las celdas (O, que también debía ser considerada en la búsqueda en ambos casos), sólo que las ponderaciones de estos últimos términos estaban relacionadas con el significado de la heurística, de modo que se ponderaría mucho la reducción de la distancia y poco la ocultación en el caso de la heurística para encontrar el camino más rápido, y mucho la ocultación y poco la reducción de distancia al destino en el caso del camino más seguro.

5.2.1.2. Regla de transición de estados

La regla de transición de estados, que es utilizada para guiar el movimiento de las hormigas es la típica que se utiliza en los algoritmos SCH clásicos con un solo objetivo. Ésta es:

Si ($q \leq q_0$)

$$j = \arg\max_{j \in N_i} \left\{ \tau(i,j)^\alpha \cdot \eta(i,j)^\beta \right\} \tag{5.22}$$

Sino

$$P(i,j) = \begin{cases} \dfrac{\tau(i,j)^\alpha \cdot \eta(i,j)^\beta}{\sum_{u \in N_i} \tau(i,u)^\alpha \cdot \eta(i,u)^\beta} & si\, j \in N_i \\ \\ 0 & en\, otro\, caso \end{cases} \tag{5.23}$$

Dónde nuevamente, $q_0 \in [0,1]$ es el parámetro estándar del SCH y q es un valor aleatorio en [0,1]. τ es la matriz de feromona y η es la función heurística (Ecuación 5.20). α y β son los parámetros de ponderación usuales y N_i es el conjunto de vecinos alcanzables para el nodo i.

El funcionamiento de la regla de transición es el mismo que en casos anteriores, es decir, es utilizada por cada hormiga cuando deben decidir el siguiente nodo al que pasar. Según ella, una hormiga elegirá el mejor de los vecinos (Ecuación 5.22) o uno cualquiera en función de una probabilidad que determina la Ecuación 5.23 y dependiendo del valor aleatorio q.

5.2.1.3. Función de evaluación

Esta función asigna un coste global a cada solución encontrada por una hormiga. En ella se consideran los costes tanto en recursos, como en salud, así como la visibilidad de los nodos (celdas) del camino, por las mismas razones expuestas en el caso bi-objetivo. Su expresión es:

$$F(C_{sol}) = \sum_{n \in C_{sol}} [R(n-1,n) + S(n-1,n) + \omega_F^o \cdot (1 - O(n))] \tag{5.24}$$

Dónde C_{sol} es el camino solución a evaluar y ω_F^o es el peso que asigna la importancia a la visibilidad de las celdas del camino y que se calcula como el valor medio de los pesos que se utilizaban en la implementación bi-criterio (Ecuación 5.20), es decir $\omega_F^o = \frac{\omega_{F,r}^o + \omega_{F,s}^o}{2}$. Los otros términos son los mismos que en la Ecuación 5.20.

5.2.1.4. Actualización de feromona

Nuevamente, al ser mono-CHAC un SCH, se tendrán dos niveles de actualización de feromona, local y global.

Actualización local de feromona. Se hace cada vez que un nuevo nodo j es añadido al camino (posible solución) que una hormiga está construyendo. Ésta se lleva a cabo con las expresiones:

$$\tau(i,j)^t = (1-\rho) \cdot \tau(i,j)^{t-1} + \rho \cdot \tau_0 \tag{5.25}$$

Dónde t se refiere al estado actual y $t-1$ al estado anterior, $\rho \in [0,1]$ es el factor de evaporación habitual y τ_0 es la cantidad inicial de feromona en cada arco:

$$\tau_0 = \frac{1}{num\_cels \cdot (MAX_R + MAX_S)/2} \tag{5.26}$$

Siendo de nuevo, num_cels el número de celdas del escenario a resolver, MAX_R el número máximo de recursos que puede requerir atravesar una celda y MAX_S el máximo coste de salud (energía) que puede requerir pasar por una celda (en el peor de los casos). Como se puede ver, esta ecuación se ha formulado a partir de las Ecuaciones 5.13 y 5.14, pero haciendo una media de los costes máximos.

Actualización global de feromona. Ésta se realiza una vez que todas las hormigas han acabado de construir sus soluciones en una iteración. De forma que para cada arco (i, j) que pertenezca al mejor camino solución hasta el momento, se aplica la expresión:

$$\tau(i,j)^t = (1-\rho)\cdot\tau(i,j)^{t-1} + \frac{\rho}{F} \tag{5.27}$$

ρ es nuevamente el factor de evaporación de feromona y t y $t-1$ se refieren al estado de feromona actual y anterior respectivamente. F es el valor asignado por la función de evaluación al camino solución (Ecuación 5.24).

5.2.2. Pseudocódigo

El pseudocódigo para el algoritmo mono-CHAC se puede ver en los Algoritmos 6 y 7:

Algoritmo 6 mono-CHAC()

```
Inicializar_feromona(τ0) {Ecuación 5.26}
for número de iteraciones do
  for cada hormiga k do
    {Ecuaciones 5.20, 5.22, 5.23, 5.25}
    camino_completo=Construir_Solucion_Mono(k)
    if camino_completo=TRUE then
      Evaluar(k.solucion) {Ecuación 5.24}
      if k.solucion.fitness < mejor_solucion.fitness then
        mejor_solucion=k.solucion
      end if
    end if
  end for
  Actualizacion_Global_Feromona(mejor_solucion) {Ecuación 5.27}
end for
```

5.3. Algoritmo CHAC-4

La idea tras el planteamiento de este algoritmo es la de intentar proponer un método para afrontar el problema planteado desde un punto de vista multiobjetivo 'más detallado'. De modo que tras haber visto las funciones

Algoritmo 7 Construir_Solucion_Mono(a)

```
{inicializaciones de niveles de recursos y salud}
a.recursos=UNIDAD.recursos
a.salud=UNIDAD.salud
{primer nodo del camino}
nodo_actual = NODO_INICIAL
a.solucion.añadir(nodo_actual)
fin=FALSE
while fin=FALSE do
  {considera los recursos y energía restante, y los obstáculos}
  N_i=Nodos_Alcanzables(nodo_actual, a)
  {hay algún nodo alcanzable}
  if N_i ≠ [ ] then
    {se elige el siguiente nodo usando la regla de transición}
    nodo_siguiente=Seleccionar_Nodo_Siguiente(nodo_actual, N_i)
    {se añade el siguiente nodo al camino solución}
    a.solucion.añadir(nodo_siguiente)
    {se calculan los consumos}
    a.recursos=a.recursos - Coste_Recursos(nodo_actual, nodo_siguiente)
    a.salud=a.salud - Coste_Salud(nodo_actual, nodo_siguiente)
    {Ecuación 5.25}
    Actualizacion_Local_Feromona(nodo_actual, nodo_siguiente)
    {la hormiga 'se mueve' al siguiente nodo}
    nodo_actual=nodo_siguiente
    {se comprueba si se ha llegado al final}
    if nodo_siguiente=NODO_DESTINO then
      fin=TRUE
      camino_completo=TRUE
    end if
  else
    fin=TRUE
    camino_completo=FALSE
  end if
end while
return(camino_completo)
```

heurísticas que se aplicaron en CHAC (Sección 5.1.1.1), se podría interpretar que el número de objetivos a considerar para resolver el problema podría ser cuatro, ya que, si bien existen dos objetivos principales a minimizar (el consumo de recursos y el consumo de salud), cada uno de ellos consta de dos subobjetivos (ponderados con los pesos deseados si alguno debe prevalecer sobre otro), es decir:

$$\begin{aligned} Obj_R &= \omega_r \cdot R + \omega_d \cdot d \\ Obj_S &= \omega_s \cdot S + \omega_o \cdot O \end{aligned} \tag{5.28}$$

Por tanto, el objetivo de la *rapidez* (o consumo mínimo de recursos) depende tanto de la minimización del consumo de recursos, como de la minimización de la distancia media al destino (esto es, que las celdas se acerquen al destino lo más rápidamente posible). De la misma forma, el objetivo de la *seguridad* (o consumo mínimo de salud/energía) depende tanto de la minimización del consumo de salud, como del nivel de ocultación de las celdas del camino (las celdas ocultas son más seguras y garantizan menos daños en un futuro).

Podría entenderse como que esta formulación en dos objetivos no es 'pura', en el sentido de que cada objetivo se define como una función agregativa compuesta por dos términos. De modo que se ha replanteado el problema definiendo cuatro objetivos en la resolución del mismo, si bien se podrían agrupar en dos grandes objetivos que son los postulados desde un principio.

Por tanto, los objetivos a minimizar pasan a ser:

- *consumo de recursos*: el camino deberá discurrir por celdas que requieran el menor consumo posible de los recursos de la unidad.
- *distancia media al destino*: las celdas del camino deberán estar cada vez más cerca del destino. Los caminos con menor media tendrán celdas más próximas al destino en general, lo que significará que serán más directos.
- *consumo de salud*: las celdas a atravesar en el camino deberían suponer el menor consumo de salud posible para la unidad, evitando para ello las más penalizadas tanto en dicho consumo, como en letalidad por impacto de armas enemigas.
- *visibilidad*: el camino deberá contener el mayor número de celdas ocultas al enemigo (no visibles), pues de esa forma, se garantiza la seguridad de la unidad respecto a posibles ataques.

Considerando que los dos primeros se decantan por la búsqueda de caminos rápidos y los dos últimos por la búsqueda de caminos seguros.

Estos objetivos son independientes entre si (aunque estén relacionados dos a dos), pero el hecho de que se consuman menos recursos no tiene por qué significar que la distancia media al destino sea cada vez menor, ni viceversa.Ya que, por ejemplo, el consumo de recursos depende también del tipo de celda a atravesar y la distancia no. Por otra parte, el consumo de salud y la visibilidad son completamente independientes (una celda tiene asociado un consumo de recursos y una visibilidad para los enemigos sin relación alguna entre dichos valores), aunque ambos están relacionados con la seguridad de la unidad.

A continuación se describirá el algoritmo propuesto, poniendo especial atención de nuevo a las funciones heurísticas, reglas de transición, actualización de feromona y funciones de evaluación, como elementos fundamentales de un algoritmo de OCH.

5.3.1. Descripción detallada del algoritmo

CHAC-4 es la adaptación del algoritmo CHAC (bi-criterio) al replanteamiento del problema de búsqueda de camino óptimo de una unidad militar en el campo de batalla considerando cuatro objetivos.

Se trata, al igual que en los casos precedentes de un SCH adaptado para tratar varios objetivos, por tanto, para resolver este problema, se deberá transformar el campo de batalla en un grafo dónde las celdas del grid hexagonal sean nodos y estén conectados con sus vecinos mediante arcos. Dado que se considerarán cuatro objetivos, dichos arcos tendrán asociados cuatro pesos que modelarán el consumo de recursos, salud, distancia al punto de destino y su visibilidad (respecto a los enemigos conocidos o la media de las visibilidades de sus vecinos en un radio, si no hay enemigos conocidos)

Las propiedades y el funcionamiento del algoritmo son similares a las de los casos anteriores y su mayor diferencia radica en el hecho de que se considerarán cuatro funciones heurísticas, cuatro funciones de evaluación y cuatro matrices de feromona, una de cada por cada uno de los objetivos a minimizar.

Al ser un algoritmo multiobjetivo, habrá que considerar la existencia de un Frente de Pareto con las mejores soluciones (las no dominadas en cuatro objetivos). Además, al igual que en el caso de CHAC, se han implementado dos posibles reglas de transición, una 'clásica' y otra basada en el concepto

de dominancia.

5.3.1.1. Funciones heurísticas

Como ya se ha comentado, CHAC-4 considerará cuatro funciones heurísticas, una por cada objetivo:

$$\eta_r(i,j) = \frac{1}{R(i,j)} \tag{5.29}$$

$$\eta_d(i,j) = \frac{1}{d(j,T)} \tag{5.30}$$

$$\eta_s(i,j) = \frac{1}{S(i,j)} \tag{5.31}$$

$$\eta_v(i,j) = O(j) \tag{5.32}$$

En la Ecuación 5.29, R será nuevamente el consumo de recursos cuando se pasa del nodo i al j, por su parte d (Ecuación 5.30) es la distancia Euclídea entre dos nodos, siendo T el nodo destino del problema. En la Ecuación 5.31, S es el coste en salud que implica pasar de un nodo i a otro j y finalmente, O (en la Ecuación 5.32) es la ocultación del nodo j al que se pasaría y es un número (entre 0 y 1) que valora la ocultación de dicho nodo, como ya se comentó en la Sección 5.2.1.1, siendo 1 cuando la celda que evalúa esté oculta a todos los enemigos (o a todas las celdas en un radio, si no hay enemigo conocido) y cuyo valor decrece exponencialmente cuando es vista.

Se habla de ocultación (en lugar de visibilidad) en la función heurística porque las heurísticas deben tener mayor valor para señalar los movimientos más deseables entre nodos y, a mayor ocultación, más deseable sería moverse al nodo j que valora la heurística. En cambio si se considerase la visibilidad, un valor grande para la heurística se traduciría en una gran visibilidad, lo cual no sería deseable en este problema.

Siguiendo el mismo planteamiento, las otras funciones heurísticas toman valores inversos a lo que se pretende minimizar para tomar valores mayores cuando menores sean los costes (R y S) o la distancia al destino.

Es importante notar además, que ya no se tienen pesos para ponderar los términos, puesto que ahora éstos no se combinan (son independientes) y se optimizan por separado.

5.3.1.2. Reglas de transición de estados

Las reglas de transición de estados (RTE), como en los algoritmos precedentes, son utilizadas por las hormigas en la construcción del camino solución (en cada iteración). Las usan para decidir el siguiente nodo al que pasar cuando están situadas en uno en particular, es decir, guían la búsqueda.

En este caso, se han implementado nuevamente dos reglas diferentes, las cuales utilizan información de los cuatro objetivos, si bien la primera de ellas combina esta información y la segunda la considera como independiente, siguiendo un enfoque multiobjetivo más 'puro' (se basa en el concepto de dominancia).

Se define en primer lugar lo que hemos dado a llamar *término de cada objetivo* y que se expresa como:

$$T_x(i,j) = \tau_x(i,j)^{\alpha} \cdot \eta_x(i,j)^{\beta} \tag{5.33}$$

Considerando que $x = r, d, s, v$ (según la inicial de cada uno de los objetivos). τ_x es la matriz de feromona correspondiente, y cada η_x es la respectiva función heurística (Ecuaciones 5.29 a 5.32). α y β son los clásicos parámetros (en los algoritmos OCH) que fijan la importancia relativa de la feromona y la información heurística respectivamente.

Regla de transición combinada para cuatro objetivos (RTC-4). Esta regla combinará la información relativa a los cuatro objetivos, tanto heurística, como memorística (feromona) para asignar una probabilidad a cada nodo vecino alcanzable desde la celda en la que se encuentre la hormiga. Al igual que los algoritmos precedentes, hará uso del parámetro estándar q_0 para elegir directamente el nodo con mayor probabilidad o elegirlo mediante una ruleta de selección basada en las probabilidades de cada nodo. La regla es:

Si $(q \leq q_0)$

$$j = \arg\max_{j \in N_i} \left\{ T_r(i,j)^{\lambda} \cdot T_d(i,j)^{\lambda} \cdot T_s(i,j)^{(1-\lambda)} \cdot T_v(i,j)^{(1-\lambda)} \right\} \tag{5.34}$$

Sino

$$P(i,j) = \begin{cases} \dfrac{T_r(i,j)^{\lambda} \cdot T_d(i,j)^{\lambda} \cdot T_s(i,j)^{(1-\lambda)} \cdot T_v(i,j)^{(1-\lambda)}}{\sum_{u \in N_i} T_r(i,u)^{\lambda} \cdot T_d(i,u)^{\lambda} \cdot T_s(i,u)^{(1-\lambda)} \cdot T_v(i,u)^{(1-\lambda)}} & si\ j \in N_i \\ 0 & en\ otro\ caso \end{cases} \tag{5.35}$$

Dónde nuevamente, q_0 es el parámetro estándar de SCH, q es un número aleatorio en [0,1] y N_i es el vecindario de nodos alcanzables para i.

$\lambda \in (0,1)$ es, al igual que en los algoritmos previos (CHAC, por ejemplo), un parámetro definido por el usuario para fijar la importancia relativa de los objetivos en la búsqueda del camino. Esto es, con él se decide si los caminos a buscar deberán tender a ser rápidos o seguros (objetivos planteados desde un principio). En este caso, se tienen cuatro objetivos, pero como ya se comentó, dos a dos están relacionados con la rapidez (coste en recursos y distancia media al destino) y la seguridad (coste en salud y visibilidad para los enemigos). Por tanto, y como se puede ver en la fórmula se usa la ponderación por λ para los objetivos relacionados con la rapidez y $1-\lambda$ para los relacionados con la seguridad.

De modo que, si por ejemplo, el usuario decidiese buscar caminos rápidos (poco consumo en recursos y distancia media al destino corta), λ tomaría un valor cercano a 1. En el caso contrario, al buscar caminos en los que prime la seguridad, λ debería ser cercano a 0. Como ya se comentó en el algoritmo CHAC (Apartado 5.1) dicho valor permanece constante para todas las hormigas y durante toda la ejecución del algoritmo, de forma que la búsqueda se centra en una zona concreta del espacio de soluciones (que dependerá del valor de dicho parámetro).

El funcionamiento de la regla es el mismo que en casos anteriores.

Regla de transición basada en dominancia para cuatro objetivos (RTD-4). Esta regla utilizará los términos relativos a cada objetivo de manera independiente del resto, es decir, no los combinará. De hecho, hará uso de ellos atendiendo al concepto de *dominancia*, comentado en el Capítulo 4 (Ecuación 4.2), el cual está estrechamente relacionado con los problemas multiobjetivo y es ampliamente utilizado por los algoritmos implementados para resolver este tipo de problemas (como se puede consultar en el Apartado 4.2).

Como se puede ver, en la dominancia se comparan distintas soluciones que satisfacen todos los objetivos, haciendo uso de funciones de coste (una por cada objetivo). Por tanto, en este caso, se considerarán como funciones de coste los *términos* definidos según la Ecuación 5.33. De este modo, se tendrán cuatro funciones de coste, las cuales se definirán como:

$$C_r(i,j) = T_r(i,j) \tag{5.36}$$

$$C_d(i,j) = T_d(i,j) \tag{5.37}$$

$$C_e(i,j) = T_e(i,j) \tag{5.38}$$

$$C_v(i,j) = T_v(i,j) \tag{5.39}$$

Considerando la dominancia, pero tendiendo a la maximización en lugar de a la minimización (como se comentó en la Ecuación 4.2), se tendrá para dos vectores a y b de cuatro valores, uno por objetivo:

$$\begin{array}{l} a \succ b \;\; sii: \\ \forall x \in [r,d,s,v] \mid C_x(a) \geq C_x(b) \;\; \wedge \;\; \exists y \in [r,d,s,v] \mid C_y(a) > C_y(b) \end{array} \tag{5.40}$$

De modo que, aplicando este último concepto, se ha definido una función, que considera como vectores de cuatro valores a comparar los arcos del grafo, y opera teniendo en cuenta las funciones de coste definidas en las Ecuaciones 5.36 a 5.39. Ésta función es:

$$D(i,j,u) = \begin{cases} 1 & si\ (i,j) \succ (i,u) \\ \\ 0 & en\ otro\ caso \end{cases} \tag{5.41}$$

Es decir, esta función comprueba si el arco (i,j) domina al arco (i,u), considerando las cuatro funciones de coste.

Por último, la regla de transición basada en dominancia se define (a partir de esta última función):

Si $(q \leq q_0)$

$$j = \arg\max_{j \in N_i} \left\{ \sum_{u \in N_i} D(i,j,u) \quad \forall j \neq u \right\} \tag{5.42}$$

Sino

$$P(i,j) = \begin{cases} \dfrac{\left(\sum_{u \in N_i} D(i,j,u)\right) + 1}{\sum_{k \in N_i} \left(\left(\sum_{u \in N_i} D(i,k,u)\right) + 1\right)} & si\, j \in N_i \wedge j \neq u \wedge k \neq u \\ 0 & en\, otro\, caso \end{cases} \tag{5.43}$$

Se puede notar que, tanto la función D, como la definición de la regla de transición es similar a las utilizadas en CHAC (Apartado 5.1.1.2-(RTD)).

N_i es nuevamente el vecindario alcanzable de i. Esta regla elige el siguiente nodo j en el camino (estando situada en el nodo i) considerando el número de vecinos dominados por cada uno de los nodos candidatos.

Al ser un SCH, se utilizará nuevamente el parámetro estándar q_0 y el valor aleatorio q, de forma que $q \leq q_0$, se elige el nodo que domine a más vecinos de i como siguiente (Ecuación 5.34). En caso contrario se usa una ruleta de selección considerando como probabilidad de cada nodo candidato, el número de nodos del vecindario a los que domina (Ecuación 5.43). De nuevo, en esta última ecuación se añade un 1 en el numerador y en el denominador para evitar una probabilidad 0 si un nodo no domina a ningún otro.

Al igual que en el caso de CHAC, esta regla utiliza conceptos de los problemas (y algoritmos) multiobjetivo a la hora de comparar nodos sin utilizar una expresión agregativa (que combine información de todos los objetivos), como es el caso de la RTC-4 descrita anteriormente. Desde un punto de vista meramente multiobjetivo, se podría considerar como más 'pura' (o más correcta), si bien, esta regla resultará añadirá un componente más explorativo al algoritmo, pues habrá muchas ocasiones en las que varios nodos tendrán la misma probabilidad asociada (dominarán al mismo número de vecinos), ya que las posibilidades de dominación se encuentran entre 0 y 6 (máximo número de vecinos alcanzables) y serán números enteros. De forma que se deberán ajustar los parámetros del algoritmo a fin de que éste tienda más a la explotación que en los casos precedentes y así equilibrar un poco la búsqueda.

5.3.1.3. Funciones de evaluación

Al igual que los demás términos del algoritmo, se han definido cuatro funciones de evaluación (una por cada objetivo), que asignan un coste global a cada una de las soluciones que encontradas por las hormigas. Sus expresiones son:

$$F_r(C_{sol}) = \sum_{n \in C_{sol}} [R(n-1, n)] \tag{5.44}$$

$$F_d(C_{sol}) = \sum_{n \in C_{sol}} [d(n, T)/N] \tag{5.45}$$

$$F_s(C_{sol}) = \sum_{n \in C_{sol}} [S(n-1, n)] \tag{5.46}$$

$$F_v(C_{sol}) = \sum_{n \in C_{sol}} [(1 - O(n))] \tag{5.47}$$

Considerando que C_{sol} es el camino solución a evaluar, n es un nodo de dicha ruta y N es el número total de nodos del camino que se está evaluando. El resto de términos son los mismos que se comentaron en las funciones heurísticas (Ecuaciones 5.29 a 5.32).

En este caso no se ha considerado ningún peso, de forma que todos los objetivos tendrán la misma importancia, a priori, en la búsqueda. Aún así, el usuario podrá determinar la prioridad de la seguridad y rapidez global.

5.3.1.4. Actualización de feromona

Dado que CHAC-4 es un SCH, existirán dos niveles de actualización de feromona: local y global. En ellos se actualizarán las cuatro matrices de feromona cada vez.

Actualización local de feromona. Se llevará a cabo cada vez que una hormiga añada un nuevo nodo al camino que esté construyendo, es decir, cada vez que se mueva al vecino que haya seleccionado como siguiente de aquel en el que esté situada, mediante la RTE. Por tanto, la actualización se realizará sobre el nuevo arco añadido a la solución (entre el nodo en el que estaba situada la hormiga y el que se ha elegido).

El objetivo de esta actualización es realizar una pequeña evaporación, y menor aporte, que 'desanime' un poco a las demás hormigas a seguir su

mismo camino y así explorar otras posibilidades. Es decir, se aplica para aumentar un poco el componente explorativo en el algoritmo.

Las fórmulas de actualización local de feromona en cada matriz son:

$$\tau_r(i,j)^t = (1-\rho)\cdot\tau_r(i,j)^{t-1} + \rho\cdot\tau_{0,r} \tag{5.48}$$

$$\tau_d(i,j)^t = (1-\rho)\cdot\tau_d(i,j)^{t-1} + \rho\cdot\tau_{0,d} \tag{5.49}$$

$$\tau_s(i,j)^t = (1-\rho)\cdot\tau_s(i,j)^{t-1} + \rho\cdot\tau_{0,s} \tag{5.50}$$

$$\tau_v(i,j)^t = (1-\rho)\cdot\tau_v(i,j)^{t-1} + \rho\cdot\tau_{0,v} \tag{5.51}$$

Considerando que t se refiere al valor actualizado de la feromona y $t-1$ al valor anterior, siendo $\rho \in [0,1]$ el clásico factor de evaporación, y los $\tau_{0,r}$, $\tau_{0,r}$, $\tau_{0,r}$ y $\tau_{0,r}$ los niveles iniciales de feromona en cada arco del grafo, para cada uno de los objetivos respectivamente (cantidades muy pequeñas que actúan como aporte). Éstos se definen como:

$$\tau_{0,r} = \frac{1}{(num\_cels \cdot MAX_R)} \tag{5.52}$$

$$\tau_{0,d} = \frac{1}{(num\_cels \cdot MAX_D)} \tag{5.53}$$

$$\tau_{0,s} = \frac{1}{(num\_cels \cdot MAX_S)} \tag{5.54}$$

$$\tau_{0,v} = \frac{1}{(num\_cels \cdot MAX_V)} \tag{5.55}$$

Considerando num_cels como el número de celdas del mapa en el que se quiere encontrar el camino óptimo, MAX_R y MAX_S como el máximo consumo de recursos y salud que requerirá atravesar una celda en el peor de los casos, MAX_D como la distancia máxima entre dos celdas cualesquiera (tamaño de la diagonal del mapa) y MAX_V como el máximo valor para la función de ocultación (O), es decir, 1.

Estas cantidades podrán salir diferentes, pero no distarán mucho entre si. En cualquier caso esto sólo influiría sobre la RTC, ya que es la única que combina los valores de las feromonas de los diferentes objetivos, pero en la práctica su influencia es ínfima y estos valores no repercuten en el comportamiento esperado durante la búsqueda según los diferentes objetivos. En el caso de la RTD, su influencia es completamente nula, ya que dichos valores nunca se llegan a combinar y simplemente se comparan por separado.

Actualización global de feromona. Esta actualización la realizarán las hormigas, una vez que todas hayan concluido la construcción de su camino solución en una iteración concreta. Únicamente la llevarán a cabo aquellas hormigas cuyas soluciones estén incluidas entre las no dominadas (dentro del FP), por lo que sólo se aplicará a los arcos que pertenezcan a dichas soluciones.

En este caso, con esta acción se pretende potenciar los mejores caminos encontrados hasta el momento, para que las hormigas consideren sus arcos como parte de futuras buenas soluciones. Esto es, se aumenta el componente de explotación en el algoritmo.

Las fórmulas de la actualización global sobre las cuatro matrices de feromona son:

$$\tau_r(i,j)^t = (1-\rho)\cdot\tau_r(i,j)^{t-1} + \rho/F_r \tag{5.56}$$

$$\tau_d(i,j)^t = (1-\rho)\cdot\tau_d(i,j)^{t-1} + \rho/F_d \tag{5.57}$$

$$\tau_s(i,j)^t = (1-\rho)\cdot\tau_s(i,j)^{t-1} + \rho/F_s \tag{5.58}$$

$$\tau_v(i,j)^t = (1-\rho)\cdot\tau_v(i,j)^{t-1} + \rho/F_v \tag{5.59}$$

Dónde t y $t-1$ se refieren nuevamente al estado actual y anterior, ρ es el mismo parámetro que en la actualización local, y dónde F_r, F_d, F_s y F_v son las funciones de evaluación correspondientes a cada objetivo (Ecuaciones 5.44 a 5.47 respectivamente).

De modo que el aporte de feromona estará en función de la calidad de la solución a la que corresponda el arco a actualizar, y será mayor para una determinada matriz de feromona cuanto mejor sea la solución considerando el objetivo correspondiente. Esto significa, como ya se comentó anteriormente, que el aporte que se hará a la feromona de cada arco para cada objetivo, será inversamente proporcional al valor de la función objetivo que se pretende optimizar.

5.3.2. Pseudocódigo

El pseudocódigo para el algoritmo CHAC-4 se puede ver en los Algoritmos descritos en 8 y 9:

Algoritmo 8 CHAC-4()

```
Inicializar_feromona(τ0,r,τ0,d,τ0,s,τ0,v) {Ecuaciones 5.52, 5.53, 5.54 y 5.55}
for número de iteraciones do
  for cada hormiga k do
    {Ecuaciones 5.29, 5.30, 5.31, 5.32, [5.34, 5.35] o [5.42, 5.43], 5.48, 5.49,
    5.50, 5.51}
    camino_completo=Construir_Solucion_4(k)
    if camino_completo=TRUE then
      Evaluar(k.solucion) {Ecuaciones 5.44, 5.45, 5.46 y 5.47}
      if dominado(Frente_Pareto, k.solucion)=FALSE then
        insertar(Frente_Pareto, k.solucion)
        eliminar_soluciones_dominadas(Frente_Pareto, k.solucion)
      end if
    end if
  end for
  Actualizacion_Global_Feromona(Frente_Pareto) {Ecuaciones 5.56, 5.57, 5.58
  y 5.59}
end for
```

5.4. Algoritmo CHAC-N

Este último algoritmo propuesto se ha creado con el fin de fijar las pautas de un nuevo método de resolución de problemas con varios objetivos, aunque centrado principalmente en problemas de búsqueda de camino óptimo, como el que se ha comentado y resuelto a lo largo de este trabajo.

CHAC-N es por tanto, un algoritmo genérico para la resolución de problemas que consideren cualquier número de funciones a minimizar. Dicho algoritmo ha sido propuesto como generalización de los anteriormente expuestos (CHAC, mono-CHAC y CHAC-4), por lo que se trata de un SCH adaptado para tratar con múltiples objetivos (en este caso N), es decir, se trata nuevamente de un algoritmo OCHMO.

Dado que es un OCH, se podrá aplicar a cualquier problema que sea transformado previamente en un grafo con pesos en los arcos. Simplemente, para poder tratar N objetivos, se tendrán N pesos asociados a cada arco, que indicarán los costes que se consumirían al atravesarlo considerando cada uno de los objetivos.

En la siguiente sección se detallarán los elementos principales del algoritmo propuesto, como son las funciones heurísticas, reglas de transición,

Algoritmo 9 Construir_Solucion_4(a)

```
{inicializaciones de niveles de recursos y salud}
a.recursos=UNIDAD.recursos
a.salud=UNIDAD.salud
{primer nodo del camino}
nodo_actual = NODO_INICIAL
a.solucion.añadir(nodo_actual)
fin=FALSE
while fin=FALSE do
  {considera los recursos y energía restante, y los obstáculos}
  N_i=Nodos_Alcanzables(nodo_actual, a)
  {hay algún nodo alcanzable}
  if N_i ≠ [ ] then
    {se elige el siguiente nodo usando una regla de transición}
    {RTC: Ecuaciones 5.34 y 5.35 o RTD: Ecuaciones 5.42 y 5.43}
    nodo_siguiente=Seleccionar_Nodo_Siguiente(nodo_actual, N_i)
    {se añade el siguiente nodo al camino solución}
    a.solucion.añadir(nodo_siguiente)
    {se calculan los consumos}
    a.recursos=a.recursos - Coste_Recursos(nodo_actual, nodo_siguiente)
    a.salud=a.salud - Coste_Salud(nodo_actual, nodo_siguiente)
    {Ecuaciones 5.48, 5.49, 5.50 y 5.51}
    Actualizacion_Local_Feromona(nodo_actual, nodo_siguiente)
    {la hormiga 'se mueve' al siguiente nodo}
    nodo_actual=nodo_siguiente
    {se comprueba si se ha llegado al final}
    if nodo_siguiente=NODO_DESTINO then
      fin=TRUE
      camino_completo=TRUE
    end if
  else
    fin=TRUE
    camino_completo=FALSE
  end if
end while
return(camino_completo)
```

actualización de feromona y funciones de evaluación (fundamentales en un algoritmo de OCH).

5.4.1. Descripción detallada del algoritmo

Al igual que los anteriores, el algoritmo que implementa CHAC-N es constructivo, por lo que cada hormiga construye una solución completa en cada iteración moviéndose por el grafo. Teniendo en cuenta que algunas veces no será posible, ya que puede ser que existan ciertas restricciones que impidan su completa.

Para guiar el movimiento durante la construcción, el algoritmo usa dos tipos de información: los *rastros de feromona* y la *información heurística* que son combinadas en muchos casos.

CHAC-N considera N *matrices de feromonas*, una por objetivo y N *funciones heurísticas* (que también son matrices calculadas al principio del proceso). Además, utiliza *una única colonia de hormigas* y hace uso del parámetro estándar de los SCH q_0, para tener un mayor control en el equilibrio entre exploración y explotación.

Nuevamente, se han planteado dos reglas de transición de estados diferentes, una basada en la combinación de la información de todos los objetivos y la otra basada en dominancia de vecinos considerando la información de cada objetivo por separado.

Las actualizaciones de feromona se siguen haciendo a dos niveles (local y global) y siguen el modelo propuesto en los algoritmos comentados en las secciones anteriores.

5.4.1.1. Funciones heurísticas

Estas funciones tratan de guiar la búsqueda entre el punto origen y el punto destino, considerando los factores más importantes para cada objetivo. Para el arco *(i,j)* habría N heurísticas, las cuales se podrían formular como:

$$\eta_n(i,j) = f(C_n(i,j)) \qquad \forall n = 1..N \tag{5.60}$$

Cada una de las funciones heurísticas estará definida para cada arco (i,j) dentro del grafo, y tomará un valor directamente proporcional a la conveniencia de optar por ese determinado arco en la construcción de un camino, es decir, dicha función tendrá un valor mayor para los arcos más deseables cuando se pretende minimizar un objetivo concreto. Cada heurística estará en

función de un coste, tomado como referencia y notado por C_n en la ecuación, por lo que su valor será inversamente proporcional a dicho coste.

5.4.1.2. Reglas generales de transición de estados

Las reglas que se utilizan para determinar el siguiente nodo j al que pasar cuando se está construyendo un camino y se está situado en el nodo i (ver Apartado 3.3) son nuevamente dos. Ambas son reglas multiobjetivo, por lo que el uso de una u otra dará como resultado dos algoritmos diferentes. Se trata de formulaciones genéricas de las comentadas para los algoritmos precedentes.

Regla general de Transición Combinada (RTC). Al igual que en los casos anteriores, se trata de la regla proporcional pseudo-aleatoria que se usa normalmente en los SCH, pero adaptada en esta ocasión para trabajar con N objetivos, combinando la información heurística y la feromona relativa a cada uno de ellos. Si bien en este caso se considerará que todos los objetivos tienen la misma importancia. La regla es:

Si $(q \leq q_0)$

$$j = \arg\max_{j \in N_i} \left\{ \prod_{n=1}^{N} \tau_n(i,j)^{\alpha} \cdot \eta_n(i,j)^{\beta} \right\} \tag{5.61}$$

Sino

$$P(i,j) = \begin{cases} \dfrac{\prod_{n=1}^{N} \tau_n(i,j)^{\alpha} \cdot \eta_n(i,j)^{\beta}}{\sum_{u \in N_i} \prod_{n=1}^{N} \tau_n(i,u)^{\alpha} \cdot \eta_n(i,u)^{\beta}} & si\, j \in N_i \\ 0 & en\, otro\, caso \end{cases} \tag{5.62}$$

Dónde nuevamente $q_0 \in [0,1]$ es el parámetro estándar del SCH y q es un valor aleatorio en [0,1]. Los τ_n son las matrices de feromona de los objetivos y los η_n son las funciones heurísticas de dichos objetivos (Ecuación 5.60). Todas estas matrices tienen un valor para cada arco (i,j). α y β son los parámetros de ponderación usuales y N_i es el conjunto de vecinos alcanzables (aquellos que cumplen las restricciones) para el nodo i.

Dado que todos los objetivos tienen la misma prioridad, el parámetro λ ya no se tiene en cuenta (aunque se podría añadir uno para asignar importancia relativa a cada objetivo, si se desease).

El funcionamiento de la regla es igual que en los casos anteriores: cuando una hormiga, que está construyendo una solución, se encuentra situada en el nodo i, si $q \leq q_0$, el mejor vecino j (el que mayor valor tenga para la expresión de la Ecuación 5.61) es seleccionado como siguiente. En otro caso, el algoritmo decide a qué nodo pasar mediante una ruleta de selección de probabilidades, considerando $P(i,j)$ como probabilidad para cada nodo alcanzable j (Ecuación 5.62).

Regla general de Transición basada en Dominancia (RTD). Esta regla es similar a la propuesta en las Secciones 5.1.1.2 y 5.3.1.2, por lo que también se basa en el concepto de dominancia visto en el Apartado 4.1 (Ecuación 4.2), pero considerando como funciones objetivo N funciones de coste. Dichas funciones trabajan con arcos que son los que tienen asignados la información heurística y memorística y la combinan:

$$C_n(i,j) = \tau_n(i,j)^\alpha \cdot \eta_n(i,j)^\beta \qquad \forall n = 1,..,N \tag{5.63}$$

Si se tiende a la maximización en lugar de a la minimización (como se comentó en la Ecuación 4.2), se tendría, para dos vectores a y b con N valores, uno por objetivo:

$$\begin{array}{l} a \succ b \;\; sii: \\ \forall x = 1,..,N \mid C_x(a) \geq C_x(b) \;\; \wedge \;\; \exists y = 1,..,N \mid C_y(a) > C_y(b) \end{array} \tag{5.64}$$

De modo que se ha definido una función que hace uso de este último concepto, considerando los arcos del grafo (como vectores de N valores a comparar) y operando teniendo en cuenta las funciones de coste definidas en la Ecuación anterior (5.63).

$$D(i,j,u) = \begin{cases} 1 & si\ (i,j) \succ (i,u) \\ \\ 0 & en\ otro\ caso \end{cases} \tag{5.65}$$

Por último, la regla general de transición basada en dominancia será:

Si $(q \leq q_0)$

$$j = \arg \max_{j \in N_i} \left\{ \sum_{u \in N_i} D(i,j,u) \quad \forall j \neq u \right\} \tag{5.66}$$

Sino

$$P(i,j) = \begin{cases} \dfrac{\left(\sum_{u \in N_i} D(i,j,u)\right) + 1}{\sum_{k \in N_i} \left(\left(\sum_{u \in N_i} D(i,k,u)\right) + 1\right)} & si\, j \in N_i \wedge j \neq u \wedge k \neq u \\ 0 & en\, otro\, caso \end{cases} \tag{5.67}$$

Dónde N_i es nuevamente el vecindario de i (nodos alcanzables desde él). Esta regla elige el siguiente nodo j en el camino (cuando la hormiga está situada en el nodo i) considerando el número de vecinos dominados por cada uno de los arcos que se pueden formar desde i con sus vecinos. De modo que si $q \leq q_0$, se elige como siguiente, el nodo cuyo arco con i domine a más (arcos) vecinos de i (Ecuación 5.66). En caso contrario se usa una ruleta de selección considerando el número de (arcos) vecinos de i que domina cada arco como probabilidad asociada (Ecuación 5.67). En esta última ecuación se añade un 1 en el numerador y en el denominador para evitar una probabilidad 0 si un determinado arco no domina a ningún otro o una indeterminación si ningún arco domina a ningún otro.

5.4.1.3. Funciones de evaluación

Son las funciones que asignan un coste global a cada solución encontrada por una hormiga. Consideran todos los costes de pasar a cada nodo (celda) en el camino solución, de modo que se tendrán N funciones de evaluación (una por objetivo):

$$F_n(C_{sol}) = \sum_{c \in C_{sol}} C_n(c-1, c) \qquad \forall n = 1, .., N \tag{5.68}$$

Dónde C_{sol} es el camino solución que se va a evaluar y $C_n(i,j)$ es el coste de pasar del nodo i al nodo j considerando el objetivo n.

5.4.1.4. Actualización de feromona

Dado que CHAC-N es un SCH, hay dos niveles de actualización de feromona, local y global. En cada uno de ellos se actualizan N matrices de feromona.

Actualización local de feromona. Como ya se vio anteriormente, se lleva a cabo cada vez que un nuevo nodo j es añadido al camino (posible solución) que una hormiga está construyendo; por tanto, ésta se aplica al nuevo arco incluido en el camino, (i, j).

El objetivo de esta actualización es disuadir a las demás hormigas de que sigan el mismo camino que la que realiza la actualización (mediante una evaporación y un muy pequeño aporte). Con ello se pretende que las demás busquen en otras zonas del espacio de soluciones (se aumenta la exploración).

Ésta se lleva a cabo con la expresión:

$$\tau_n(i,j)^t = (1-\rho)\cdot\tau_n(i,j)^{t-1} + \rho\cdot\tau_{0,n} \qquad \forall n = 1,..,N \tag{5.69}$$

Señalando con t el nuevo valor para la feromona y con $t-1$ el anterior. Asimismo $\rho \in [0,1]$ es el factor de evaporación común y cada $\tau_{0,n}$ es la cantidad inicial de feromona en cada arco para cada objetivo, la cual constituye el aporte mínimo que se pretende hacer sobre la feromona de los arcos ya visitados por la hormiga actual. Dicha cantidad se calcula como:

$$\tau_{0,n} = \frac{1}{(num\_nodos \cdot MAX_{C,n})} \qquad \forall n = 1,..,N \tag{5.70}$$

Siendo *num_nodos* el número de nodos del grafo a resolver y $MAX_{C,n}$ el coste máximo para el objetivo n que puede requerir atravesar una celda (en el peor de los casos).

Actualización global de feromona. Nuevamente, esta actualización se hará considerando únicamente las soluciones dentro del Frente de Pareto (soluciones no dominadas), es decir, se aplicará a los arcos que formen parte de dichos caminos. Por tanto, ésta se llevará a cabo una vez que todas las hormigas hayan acabado de construir sus soluciones y se hayan valorado las posibles dominancias entre ellas.

El objetivo que persigue esta actualización es el de potenciar los buenos caminos para que las hormigas de las siguientes iteraciones consideren sus

arcos como buenas opciones en la búsqueda de soluciones óptimas (aumento de la explotación).

La expresión utilizada en esta actualización es:

$$\tau_n(i,j)^t = (1-\rho)\cdot\tau_n(i,j)^{t-1} + \rho/F_n \qquad \forall n = 1,..,N \tag{5.71}$$

Nuevamente se ha señalado la nueva feromona con t y la anterior con $t-1$. ρ es el mismo parámetro que se usó para la actualización local y F_n es el coste global asociado a la solución, calculad usando la Ecuación 5.68.

Como se puede observar, el aporte de feromona será directamente proporcional a la calidad de la solución a la que corresponda el arco a actualizar, es decir, inversamente proporcional a la función que se pretende minimizar.

5.4.2. Pseudocódigo

El pseudocódigo para el algoritmo CHAC-N se puede ver en los Algoritmos 10 y 11:

Algoritmo 10 CHAC-N()

```
Inicializar_feromona(τ0,1,..,τ0,N) {Ecuación 5.70}
for número de iteraciones do
  for cada hormiga k do
    {Ecuaciones 5.60, [5.61, 5.62] o [5.66, 5.67], 5.69}
    camino_completo=Construir_Solucion_N(k)
    if camino_completo=TRUE then
      Evaluar(k.solucion) {Ecuación 5.68}
      if dominado(Frente_Pareto, k.solucion)=FALSE then
        insertar(Frente_Pareto, k.solucion)
        eliminar_soluciones_dominadas(Frente_Pareto, k.solucion)
      end if
    end if
  end for
  Actualizacion_Global_Feromona(Frente_Pareto) {Ecuación 5.71}
end for
```

Algoritmo 11 Construir_Solucion_N(a)

```
{inicializaciones de niveles de recursos y salud}
a.recursos=UNIDAD.recursos
a.salud=UNIDAD.salud
{primer nodo del camino}
nodo_actual = NODO_INICIAL
a.solucion.añadir(nodo_actual)
fin=FALSE
while fin=FALSE do
  {considera las restricciones}
  N_i=Nodos_Alcanzables(nodo_actual, a)
  {hay algún nodo alcanzable}
  if N_i ≠ [ ] then
    {se elige el siguiente nodo usando una regla de transición}
    {RTC: Ecuaciones 5.61 y 5.62 o RTD: Ecuaciones 5.66 y 5.67}
    nodo_siguiente=Seleccionar_Nodo_Siguiente(nodo_actual, N_i)
    {se añade el siguiente nodo al camino solución}
    a.solucion.añadir(nodo_siguiente)
    {se calculan los consumos}
    a.recursos=a.recursos - Coste_Recursos(nodo_actual, nodo_siguiente)
    a.salud=a.salud - Coste_Salud(nodo_actual, nodo_siguiente)
    {Ecuación 5.69}
    Actualizacion_Local_Feromona(nodo_actual, nodo_siguiente)
    {la hormiga 'se mueve' al siguiente nodo}
    nodo_actual=nodo_siguiente
    {se comprueba si se ha llegado al final}
    if nodo_siguiente=NODO_DESTINO then
      fin=TRUE
      camino_completo=TRUE
    end if
  else
    fin=TRUE
    camino_completo=FALSE
  end if
end while
return(camino_completo)
```

Capítulo 6

Algoritmos de la bibliografía adaptados para resolver el problema

En este capítulo se describen una serie de algoritmos que se pueden encontrar en la bibliografía adaptados para la resolución del problema que nos ocupa en este trabajo.

Se trata de dos algoritmos de colonias de hormigas multiobjetivo diseñados inicialmente para la resolución de otros problemas. Dichos algoritmos sirvieron a su vez de base para el algoritmo CHAC que se presentó anteriormente. Además, también se comenta un algoritmo voraz (Greedy) de optimización y búsqueda clásico, pero adaptado a la búsqueda de soluciones en un espacio multiobjetivo.

Por tanto, los algoritmos presentados son reformulaciones de algoritmos ya existentes, hechas para afrontar el mismo problema, el cual tendrá dos objetivos, la rapidez y la seguridad. Dichos algoritmos son:

- *MOACS*: SCH propuesto inicialmente para la resolución del problema multiobjetivo del enrutamiento de vehículos con ventana de tiempo. En él, se tiene una única matriz de feromonas y dos heurísticas (una por objetivo). Su adaptación al problema que se resuelve en el presente trabajo ha supuesto la creación de nuevas funciones heurísticas y de evaluación, así como una actualización de feromona diferente a la original.

- *BiCriterionAnt*: se trata de un SH presentado inicialmente para resolver

un problema de asignación de tareas con coste variable a máquinas. Trabaja con dos matrices de feromona y dos funciones heurísticas, al igual que CHAC, pero la adaptación al problema de búsqueda de camino óptimo con dos criterios, ha significado igualmente la consideración de nuevas funciones heurísticas, de evaluación y actualización de feromona.

- *GreedyMO*: algoritmo con enfoque voraz (greedy) adaptado para la resolución del problema. Para ello, se consideran como funciones de coste las funciones heurísticas de CHAC y se aplica un criterio basado en dominancia para seleccionar el siguiente nodo en el camino solución. De esta forma se adapta además a la resolución de un PMO. El algoritmo incluye también mejoras adicionales para garantizar que siempre se encuentre solución.

Estos algoritmos nos servirán como base comparativa respecto a los propuestos en el Capítulo 5, a fin de evaluar la bondad, posibilidades y adecuación de éstos para la resolución del problema planteado.

6.1. Algoritmo MOACS

El algoritmo llamado Sistema de Colonias de Hormigas Multiobjetivo o MOACS (Multiobjective Ant Colony System en inglés) fue presentado por Barán et al. [43], como una adaptación o variación del algoritmo MACS (MultiAnt Colony System) creado por Gambardela et al. [42]. Ambos fueron ideados para resolver el problema de enrutamiento de vehículos con ventana de tiempo, la versión multiobjetivo de problema clásico [20] (comentado en la Sección 2.1.2.

La principal diferencia entre ambos es el uso de una única matriz de feromonas para ambos objetivos en el caso de MOACS, en lugar de considerar una matriz por objetivo, como hace su homónimo (y predecesor).

En este trabajo, MOACS ha sido adaptado a la resolución del problema que nos ocupa, por tanto, dicho algoritmo compartirá ciertas funciones con CHAC. En concreto, este algoritmo hará uso de las mismas funciones heurísticas que los métodos bi-objetivo propuestos, las cuales se pueden ver en las Ecuaciones 5.1 y 5.2. También compartirá las funciones de evaluación de soluciones (Ecuaciones 5.11 y 5.12).

Las diferencias aparecen en las fórmulas correspondientes a la Regla de Transición de Estados y las actualizaciones de feromona.

Regla de Transición de Estados. Esta regla es similar a la propuesta para CHAC (ver Ecuaciones 5.3 y 5.4), pero en ella se utiliza una única matriz de feromona. Por tanto, se define como:

Si ($q \leq q_0$)

$$j = \arg\max_{j \in N_i} \left\{ \tau(i,j) \cdot \eta_r(i,j)^{\beta \cdot \lambda} \cdot \eta_s(i,j)^{\beta \cdot (1-\lambda)} \right\} \tag{6.1}$$

Sino

$$P(i,j) = \begin{cases} \dfrac{\tau(i,j) \cdot \eta_r(i,j)^{\beta \cdot \lambda} \cdot \eta_s(i,j)^{\beta \cdot (1-\lambda)}}{\sum_{u \in N_i} \tau(i,u) \cdot \eta_r(i,u)^{\beta \cdot \lambda} \cdot \eta_s(i,u)^{\beta \cdot (1-\lambda)}} & si\, j \in N_i \\ 0 & en\, otro\, caso \end{cases} \tag{6.2}$$

Dónde τ y η representan nuevamente la feromona y la función heurística (una por objetivo) y el resto de parámetros se corresponden con los expuestos en las ecuaciones de reglas de transición precedentes (Por ejemplo 5.3 y 5.4).

Esta regla hace uso nuevamente del parámetro λ para establecer la importancia de los dos objetivos en la búsqueda. Dicho parámetro mantiene un valor constante durante toda la ejecución y para todas las hormigas, a diferencia de la propuesta original de Barán et al. [43], en la que el parámetro toma un valor igual a 0 para la primera hormiga y crece de manera uniforme hasta alcanzar un valor igual a 1 para la última hormiga (aplicando $\lambda_k = \frac{k-1}{m-1}$, para cada hormiga $k \in [1, m]$). La razón para hacerlo según nuestra propuesta radica en el hecho de que estos algoritmos deben resolver un problema, pero deben considerar el criterio determinado por un usuario, el cual pondera la importancia de cada objetivo, lo que limita la zona del espacio de búsqueda en la que actúa el algoritmo.

Esta regla funciona como las precedentes.

Dado que MOACS es un SCH, realizará una actualización de feromona a dos niveles: local y global.

Actualizaciones de Feromona. La *Actualización Local* sigue la expresión:

$$\tau(i,j)^t = (1-\rho)\cdot\tau(i,j)^{t-1} + \rho\cdot\tau_0 \tag{6.3}$$

Dónde se ha marcado con t el nuevo valor para la feromona y con $t-1$ el valor anterior (antes de actualizarla), y se ha considerado como cantidad inicial de feromona:

$$\tau_0 = \frac{1}{num\_cels\cdot MAX_R^{\lambda}\cdot MAX_S^{(1-\lambda)}} \tag{6.4}$$

En ambas expresiones, los parámetros son los mismos que en las Ecuaciones 5.25 y 5.26, pero ponderando mediante el parámetro λ para considerar nuevamente la importancia relativa de cada objetivo.

Al igual que en los algoritmos comentados previamente, cada vez que una hormiga h construye una solución completa, ésta es comparada con las que haya incluidas en el Frente de Pareto P hasta ese momento, para comprobar si dicha solución es no dominada (y debe incluirse en dicho frente).

Pero en este caso, existe un *mecanismo de reinicialización* para evitar estancamientos en la búsqueda (por caer cerca de óptimos locales) que impidan mejorar las soluciones. Para ello, en cada iteración, una vez que todas las hormigas han terminado su búsqueda de soluciones, se comprueba cómo ha evolucionado el Frente de Pareto (las soluciones incluidas en él), de forma que cada vez que se hayan añadido nuevas soluciones al mismo que hagan que éste mejore en media (dichas soluciones dominan y reemplazan a algunas de las existentes), se borrarán todos los rastros de feromona hasta el momento y se inicializarán a un nuevo valor de τ_0 (que depende de las soluciones en el FP). Con ello se pretende replantear la búsqueda en nuevas zonas del espacio de soluciones, aumentando la exploración, pues es posible que la búsqueda anterior estuviese centrada en una zona en la que se encontrasen las soluciones que han resultado ser dominadas en el nuevo FP.

Con este fin, el valor de τ_0 no es fijo durante la ejecución del algoritmo, sino que se va recalculando y adaptando según evoluciona el FP. Por ello, al final de cada iteración se recalcula τ_0' según la fórmula:

$$\tau_0' = \frac{1}{num\_cels\cdot \bar{R}(P)^{\lambda}\cdot \bar{S}(P)^{(1-\lambda)}} \tag{6.5}$$

Dónde $\bar{R}$ y $\bar{S}$ son respectivamente el consumo medio de recursos y salud de las soluciones (caminos) incluidas en el FP. Ambas están nuevamente ponderadas usando λ para ponderar cada uno de los objetivos.

Por lo que si $\tau_0' > \tau_0$ (el valor de feromona inicial que se tiene actualmente), quiere decir que se ha encontrado un FP mejor, todos los rastros de feromona son reinicializados considerando el nuevo valor para τ_0, es decir τ_0'. El nuevo valor de τ_0 se utilizará en la actualización local de feromona (Ecuación 6.3).

En otro caso, se sigue con la búsqueda como hasta el momento, es decir, se continúa con la explotación de la zona del espacio de soluciones en la que se encuentre el algoritmo. De modo que se aplica la *Actualización Global de Feromona* a cada solución que haya en el FP:

$$\tau(i,j) = (1-\rho) \cdot \tau(i,j) + \frac{\rho}{F_r \cdot F_s} \tag{6.6}$$

Dónde F_r y F_s son las funciones de evaluación para cada objetivo respectivamente (Ecuaciones 5.11 y 5.12).

Pseudocódigo. El pseudocódigo del algoritmo MOACS se puede ver en los Algoritmos 12 y 13.

6.2. Algoritmo BiCriterion Ant

Este algoritmo fue presentado por Iredi et al. [97] en 2001. En dicho trabajo se mostraba la resolución de un problema clásico de programación de tareas en máquinas, el SMTTP (Single Machine Total Tardiness Problem en inglés) con costes variables, mediante un MOACO que consideraba varias colonias y se mostraba como posible solución para problemas bi-criterio cuando no se puede determinar que un objetivo tenga más importancia que otro. Además, en dicho artículo se comentaba un método que resolvía el problema usando únicamente una colonia, que es precisamente el algoritmo que ha sido considerado, el *BiCriterion Ant* (BiAnt en adelante).

Dicho algoritmo es un SH adaptado para la resolución de PMOs, utiliza una matriz de feromonas y una función heurística por cada objetivo, es decir, que se tendrán dos de cada si se va a resolver un problema bi-criterio como el que nos ocupa. Por tanto, BiAnt utilizará como matrices de feromona τ_r y τ_s (una para el objetivo de la rapidez y otra para el de la seguridad), y además compartirá con CHAC las mismas funciones heurísticas (Ecuaciones 5.1 y 5.2) y funciones de evaluación (Ecuaciones 5.11 y 5.12).

De nuevo, las diferencias aparecen en la Regla de Transición de Estados y las actualizaciones de feromona.

Algoritmo 12 MOACS()

```
Inicializar_feromona(τ0) {Ecuación 6.4}
for número de iteraciones do
  for cada hormiga k do
    {Ecuaciones 5.1, 5.2, [6.1, 6.2], 6.3}
    camino_completo=Construir_Solucion_MOACS(k)
    if camino_completo=TRUE then
      Evaluar(k.solucion) {Ecuaciones 5.11 y 5.12}
      if dominado(Frente_Pareto, k.solucion)=FALSE then
        insertar(Frente_Pareto, k.solucion)
        eliminar_soluciones_dominadas(Frente_Pareto, k.solucion)
      end if
    end if
  end for
  {Recalcular en nuevo τ0 en base a las soluciones del FP}
  τ0' = 1/(num_cels · R̄(P)^λ · S̄(P)^(1−λ))
  {Comprobar si el FP ha mejorado}
  if τ0' > τ0 then
    {Reinicialización de rastros de feromona}
    τ0 = τ0'
    Inicializar_feromona(τ0)
  else
    Actualizacion_Global_Feromona(Frente_Pareto) {Ecuación 6.6}
  end if
end for
```

Regla de Transición. En este caso, sólo se utiliza la expresión del cálculo de probabilidad para el posterior uso de la ruleta de selección, es decir, al ser un SH, no se dispone del parámetro q_0. Dicha expresión es igual a la empleada en la regla de transición combinada de CHAC (Ecuación 5.4), si bien originalmente se diferenciaba en el uso que se hacía del parámetro λ. La probabilidad que determinará el funcionamiento de la regla se define como:

$$P(i,j) = \begin{cases} \dfrac{\tau_r(i,j)^{\alpha\cdot\lambda}\cdot\tau_s(i,j)^{\alpha\cdot(1-\lambda)}\cdot\eta_r(i,j)^{\beta\cdot\lambda}\cdot\eta_s(i,j)^{\beta\cdot(1-\lambda)}}{\sum_{u\in N_i}\tau_r(i,u)^{\alpha\cdot\lambda}\cdot\tau_s(i,u)^{\alpha\cdot(1-\lambda)}\cdot\eta_r(i,u)^{\beta\cdot\lambda}\cdot\eta_s(i,u)^{\beta\cdot(1-\lambda)}} & si\ j\in N_i \\ 0 & en\ otro\ caso \end{cases} \quad (6.7)$$

Algoritmo 13 Construir_Solucion_MOACS(a)

```
{inicializaciones de niveles de recursos y salud}
a.recursos=UNIDAD.recursos
a.salud=UNIDAD.salud
{primer nodo del camino}
nodo_actual = NODO_INICIAL
a.solucion.añadir(nodo_actual)
fin=FALSE
while fin=FALSE do
  {considera los recursos y energía restante, y los obstáculos}
  N_i=Nodos_Alcanzables(nodo_actual, a)
  {hay algún nodo alcanzable}
  if N_i ≠ [ ] then
    {se elige el siguiente nodo usando la regla de transición}
    {Ecuaciones 6.1 y 6.3}
    nodo_siguiente=Seleccionar_Nodo_Siguiente(nodo_actual, N_i)
    {se añade el siguiente nodo al camino solución}
    a.solucion.añadir(nodo_siguiente)
    {se calculan los consumos}
    a.recursos=a.recursos - Coste_Recursos(nodo_actual, nodo_siguiente)
    a.salud=a.salud - Coste_Salud(nodo_actual, nodo_siguiente)
    {Ecuación 6.3}
    Actualizacion_Local_Feromona(nodo_actual, nodo_siguiente)
    {la hormiga 'se mueve' al siguiente nodo}
    nodo_actual=nodo_siguiente
    {se comprueban si se ha llegado al final}
    if nodo_siguiente=NODO_DESTINO then
      fin=TRUE
      camino_completo=TRUE
    end if
  else
    fin=TRUE
    camino_completo=FALSE
  end if
end while
return(camino_completo)
```

Dónde los τ y los η representan nuevamente las matrices de feromona y las funciones heurísticas por objetivos respectivamente, y el resto de parámetros son los ya comentados en las reglas de transición precedentes (Por ejemplo las de la Ecuación 5.4).

Como se puede ver, esta regla también hace uso del parámetro λ, siendo nuevamente su valor fijo para todas las hormigas y durante toda la ejecución, a diferencia de nuevo del planteamiento original del trabajo de Iredi et al. dónde tomaba un valor para cada hormiga $k \in [1, m]$, según la fórmula $\lambda_k = \frac{k-1}{m-1}$. De esta forma todas las hormigas centran su búsqueda en la misma zona del espacio de soluciones.

El funcionamiento de la regla es sencillo, simplemente se calcula una probabilidad para cada uno de los vecinos alcanzables (notados por j) del nodo en el que se encuentra la hormiga, i. Dicha probabilidad indica lo deseable que es pasar a cada uno de esos nodos en la construcción del camino solución. Posteriormente se construye una ruleta de probabilidades asignando a cada nodo j su valor $P(i, j)$ y eligiendo al nodo siguiente a i haciendo un lanzamiento aleatorio en dicha ruleta.

Actualización de Feromona. Al principio del algoritmo, los rastros de feromona de ambos objetivos serán inicializados a unas cantidades $\tau_{0,r}$ y $\tau_{0,s}$ calculadas según las Ecuaciones 5.15 y 5.16 de CHAC.

Dado que se trata de un SH, únicamente se hará una *actualización de feromona a nivel global*, una vez que todas las hormigas hayan terminado su búsqueda particular. Dicha actualización incluirá una *evaporación* y un *aporte*.

En primer lugar se realizará la *evaporación* en todos los arcos del grafo, siguiendo el esquema del SH (y asemejándose a la naturaleza que lo inspira), para intentar aumentar el componente explorativo en el algoritmo (al borrar parte de los rastros anteriores, se podría buscar en nuevas zonas del espacio de soluciones). Dicha evaporación se aplicará a las dos matrices según las expresiones:

$$\tau_r(i, j)^t = (1 - \rho) \cdot \tau_r(i, j)^{t-1} \tag{6.8}$$

$$\tau_s(i, j)^t = (1 - \rho) \cdot \tau_s(i, j)^{t-1} \tag{6.9}$$

Señalando con t el nuevo valor para la feromona y con $t - 1$ el anterior. $\rho \in [0, 1]$ es el conocido factor de evaporación.

Como mecanismo de control, se considera un nivel mínimo de feromona, para evitar que arcos no recorridos en una parte del algoritmo pudiesen

quedar con un nivel 0 de feromona asociada, con lo que nunca serían seleccionados en el futuro.

El *aporte* lo realizarán las hormigas cuyas soluciones estén incluidas en el FP, para hacer más atractivos dichos arcos al resto de las hormigas y aumentar así el componente de explotación. De forma que para cada arco perteneciente a una de esas soluciones se hará:

$$\tau_r(i,j)^t = \tau_r(i,j)^{t-1} + 1/F_r \tag{6.10}$$

$$\tau_s(i,j)^t = \tau_s(i,j)^{t-1} + 1/F_s \tag{6.11}$$

Nuevamente t se refiere al nuevo valor para la feromona y $t-1$ al anterior, F_r y F_s son los valores obtenidos al evaluar la solución a la que pertenece el arco que se está actualizando mediante las Ecuaciones 5.11 y 5.12 respectivamente.

Como se puede ver, el aporte de feromona será proporcional a la calidad de la solución, de modo que será mayor cuanto menor sea el coste asociado a la misma en cada objetivo.

Pseudocódigo. El pseudocódigo del algoritmo BiAnt se puede ver en los Algoritmos 14 y 15.

Algoritmo 14 BiAnt()

```
Inicializar_feromona(τ0,r, τ0,s) {Ecuaciones 5.15 y 5.16}
for número de iteraciones do
  for cada hormiga h do
    {Ecuaciones 5.1, 5.2, 6.7}
    camino_completo=Construir_Solucion_BiAnt(h)
    if camino_completo=TRUE then
      Evaluar(h.solucion) {Ecuaciones 5.11 y 5.12}
      if dominado(Frente_Pareto, h.solucion)=FALSE then
        insertar(Frente_Pareto, h.solucion)
        eliminar_soluciones_dominadas(Frente_Pareto, h.solucion)
      end if
    end if
  end for
  {Actualización global de feromona}
  Evaporacion_Feromona() {Ecuaciones 6.8 y 6.9}
  Aporte_Feromona(Frente_Pareto) {Ecuaciones 6.10 y 6.11}
end for
```

Algoritmo 15 Construir_Solucion_BiAnt(a,λ_a)

```
{inicializaciones de niveles de recursos y salud}
a.recursos=UNIDAD.recursos
a.salud=UNIDAD.salud
{primer nodo del camino}
nodo_actual = NODO_INICIAL
a.solucion.añadir(nodo_actual)
fin=FALSE
while fin=FALSE do
  {considera los recursos y energía restante, y los obstáculos}
  N_i=Nodos_Alcanzables(nodo_actual, a)
  {hay algún nodo alcanzable}
  if N_i ≠ [ ] then
    {se elige el siguiente nodo usando una regla de transición}
    {Ecuación 6.7}
    nodo_siguiente=Seleccionar_Nodo_Siguiente(nodo_actual, N_i)
    {se añade el siguiente nodo al camino solución}
    a.solucion.añadir(nodo_siguiente)
    {se calculan los consumos}
    a.recursos=a.recursos - Coste_Recursos(nodo_actual, nodo_siguiente)
    a.salud=a.salud - Coste_Salud(nodo_actual, nodo_siguiente)
    {la hormiga 'se mueve' al siguiente nodo}
    nodo_actual=nodo_siguiente
    {se comprueba si se ha llegado al final}
    if nodo_siguiente=NODO_DESTINO then
      fin=TRUE
      camino_completo=TRUE
    end if
  else
    fin=TRUE
    camino_completo=FALSE
  end if
end while
return(camino_completo)
```

6.3. Algoritmo Greedy

Los algoritmos voraces o Greedy en inglés, son metaheurísticas utilizadas para la resolución de problemas de optimización, siguiendo un esquema de búsqueda paso a paso. Su máxima es seleccionar los elementos más prometedores del conjunto de candidatos en casa paso del algoritmo, hasta encontrar una solución. Este enfoque es bastante simple, por lo que muchas veces la solución obtenida no es la óptima, pero si que resultan muy rápidos y en muchos casos obtienen soluciones bastante aceptables.

En este trabajo se ha diseñado un método de resolución del problema que nos ocupa basado en una técnica greedy. Dicho algoritmo ha sido diseñado para la resolución de problemas multiobjetivo y se ha bautizado como GRMO.

A grosso modo, GRMO se limita a buscar un camino en una sola iteración. De modo que, partiendo del nodo inicial, se pretende alcanzar el nodo final simplemente eligiendo en cada paso el mejor nodo posible (de entre los vecinos).

Tanto para adaptar el algoritmo a la resolución de un PMO, como para valorar cada uno de los nodos posibles, se ha considerado un *criterio basado en la dominancia* entre vecinos (similar al utilizado en la RTD de CHAC, comentada en el apartado 5.1.1.2). Para dicho criterio se han considerado como funciones de coste las funciones heurísticas que se definieron en CHAC (Ecuaciones 5.1 y 5.2), es decir:

$$C_r(i,j) = \eta_r(i,j) \tag{6.12}$$

$$C_s(i,j) = \eta_s(i,j) \tag{6.13}$$

Esto es, se usarán las funciones heurísticas para cada objetivo, η_r y η_s. Después, y considerando la definición de dominancia vista en la Ecuación 5.7 (con las funciones de coste que se acaban de comentar), se propone como función de dominancia entre dos arcos vecinos (ambos parten del mismo nodo) la misma que se definió en la Ecuación 5.8 para CHAC (llamada D). Dicha función llamada según el formato $D(i,j,u)$ devolverá un 1 si el arco (i,j) domina al arco (i,u), siendo ambos arcos vecinos.

Considerando todo esto, se propone una función que asignará un valor a cada uno de los nodos candidatos j, cuando se esté situado en el nodo i, según la expresión:

$$V(i,j) = \sum_{u \in N_i} D(i,j,u) \quad \forall j \in N_i \;\; y \;\; j \neq u \tag{6.14}$$

Siendo nuevamente N_i el conjunto de nodos alcanzables (vecinos) para el nodo i, en el que se está actualmente. De modo que se asignará a cada nodo j que sea candidato a ser el siguiente en la construcción de la solución, un valor correspondiente al número de vecinos a los que domina (el arco (i, j) respecto a cada uno de los posibles arcos (i, u)), considerando las funciones de coste comentadas anteriormente. A continuación, se elegirá como siguiente nodo en el camino solución, aquel cuyo valor para V sea el mayor (aquel que domine a más vecinos). Dado que el número de vecinos alcanzables es pequeño (seis a lo sumo en el modelo del problema planteado), es muy probable que haya varios nodos con el mismo valor para V. En ese caso se elegiría entre aquellos con el mayor valor de forma aleatoria. Esto añade un componente estocástico que normalmente no tienen los algoritmos greedy.

Otro aspecto a tener en cuenta es que se trata de un algoritmo simple, por lo que muchas veces no conseguirá alcanzar una solución. Ya que podría entrar en bucles (se mueve entre una serie de celdas consumiendo todos los recursos o energía) o llegar a una zona rodeada de nodos no alcanzables (rodeada de obstáculos, un borde del escenario o mapa a resolver, cuyo consumo requerido en recursos o salud sea mayor del que le resta a la unidad). Para solucionar el primero de los problemas se añadió la condición de que el camino solución no pueda pasar más de una vez por la misma celda. Estos problemas se deben en parte a que el algoritmo no hace uso de información compartida o refuerzo positivo (ni negativo) como la inmensa mayoría de los algoritmos de búsqueda de camino mínimo.

Para afrontar el segundo de los problemas comentados se implementó una mejora consistente en hacer un *backtracking* si se llega a un nodo que no tiene vecinos alcanzables. Esto es, en caso de llegar a un nodo j que no tiene vecinos alcanzables, se vuelve al nodo i que le precede en el camino solución y se elige como nodo siguiente un nodo distinto al j anterior. Si se vuelve a producir la misma situación con el resto de los posibles vecinos de i, se daría un nuevo paso atrás (un nuevo backtracking) y se volvería al nodo precedente a i, intentando construir la solución con un vecino de dicho nodo. Este mecanismo se ha implementado para hacer un backtracking a tres niveles como máximo, ya que su aplicación se traduce en un aumento considerable del tiempo de ejecución del algoritmo. Aún así, su uso da muy buenos resultados, garantizando casi completamente que se encuentre una solución.

Para terminar, se ha implementado una versión del greedy que realiza la búsqueda un número de iteraciones (obteniendo una solución en cada una)

y guarda las mejores soluciones encontradas (las no dominadas), creando un Frente de Pareto, aunque muy localizado en una zona del espacio de soluciones, debido al pequeñísimo factor explorativo que tiene el algoritmo (gracias al backtracking). La obtención de varias soluciones diferentes es posible debido al componente estocástico que se ha comentado previamente.

Pseudocódigo. El pseudocódigo para el algoritmo GRMO se puede ver en los Algoritmos 16 y 17.

Algoritmo 16 GRMO()

```
for número de iteraciones do
  {inicializaciones de niveles de recursos y salud}
  sol_greedy.recursos=UNIDAD.recursos
  sol_greedy.salud=UNIDAD.salud
  {primer nodo del camino}
  nodo_actual=NODO_INICIAL
  sol_greedy.solucion.añadir(nodo_actual)
  lista_prohibidos=[ ]
  niv_backtracking=1
  {Se busca una solución}
  Construir_Solucion_GreedyMO()
  {Se evalúa la solución y se añade al FP si es no-dominada}
  Evaluar(sol_greedy.solucion) {Ecuaciones 5.11 y 5.12}
  if dominado(Frente_Pareto, sol_greedy.solucion)=FALSE then
    insertar(Frente_Pareto, sol_greedy.solucion)
    eliminar_soluciones_dominadas(Frente_Pareto, sol_greedy.solucion)
  end if
end for
```

Algoritmo 17 Construir_Solucion_GreedyMO()

```
fin=FALSE
while fin=FALSE do
  {considera los recursos y energía restante, y los obstáculos}
  N_i=Nodos_Alcanzables(nodo_actual, sol_greedy, lista_prohibidos)
  {hay algún nodo alcanzable}
  if N_i ≠ [ ] then
    {si se llega al nivel máximo de backtracking, se borra la lista de prohibidos}
    if niv_backtracking ≥ MAX_NIV_BACKTRACKING then
      lista_prohibidos=[ ]
      niv_backtracking=1
    else
      niv_backtracking=niv_backtracking+1
    end if
    {se elige el siguiente nodo usando la Ecuación 6.14 (será el mejor)}
    nodo_siguiente=Seleccionar_Nodo_Siguiente(nodo_actual, N_i)
    {se añade el siguiente nodo al camino solución}
    sol_greedy.solucion.añadir(nodo_siguiente)
    {se calculan los consumos}
    sol_greedy.recursos=sol_greedy.recursos - Coste_Recursos(nodo_actual, nodo_siguiente)
    sol_greedy.salud=sol_greedy.salud - Coste_Salud(nodo_actual, nodo_siguiente)
    {se pasa al siguiente nodo}
    nodo_actual=nodo_siguiente
    {se comprueba si se ha llegado al final}
    if nodo_siguiente=NODO_DESTINO then
      fin=TRUE
    end if
    {no hay nodos alcanzables}
  else
    {se hace backtracking: se pasa al nodo anterior en el camino y se apunta el
    nodo actual como prohibido}
    lista_prohibidos.añadir(nodo_actual)
    sol_greedy.solucion.borrar(nodo_actual)
    nodo_actual=nodo_anterior
  end if
end while
```

Capítulo 7

Resolución de escenarios reales. Análisis y justificación de resultados

El presente capítulo está dedicado a la descripción de los experimentos desarrollados a lo largo del estudio que presenta esta tesis. Dentro de dichos experimentos, se incluye la resolución de varios PCO dentro del entorno militar, modelados (y resueltos) dentro del entorno de simulación comentado en el Apéndice B, es decir, el simulador mSS.

Dichos problemas, modelados como mapas/escenarios han sido creados con la ayuda de personal militar del Mando de Adiestramiento y Doctrina (MADOC), y serán presentados en la sección siguiente, describiendo sus características y la dificultad de resolución estimada según el criterio de un capitán humano (que dirigiese la CIA).

A continuación, se describirán las configuraciones y ajustes de parámetros llevados a cabo tras un proceso de análisis previo, a fin de obtener el mejor rendimiento de los algoritmos a aplicar. Igualmente, se definirán las restricciones que caracterizan los problemas y se analizará su influencia tanto en el espacio de búsqueda, como en el funcionamiento de los algoritmos.

Posteriormente se comentarán una serie de variaciones aplicadas sobre los valores de los parámetros que se usan en algunos de los algoritmos presentados en el Capítulo 5, a fin de obtener resultados extremos para cada uno de los objetivos principales. Con estas versiones extremas, se pretende completar el abanico comparativo, al utilizarlas como base para cotejar la bondad de los resultados obtenidos por los demás algoritmos.

Tras esto, se describirán los experimentos realizados aplicando cada uno de los algoritmos presentados en los Capítulos 5 y 6, así como las versiones extremas descritas en el presente capítulo.

Finalmente, se presentarán las conclusiones preliminares alcanzadas tras valorar los resultados obtenidos por los algoritmos en los experimentos precedentes.

7.1. Definición de escenarios a resolver

En esta sección se describirán las propiedades de los escenarios en los que se experimentará, los cuales incluirán dos mapas realistas. Éstos serán modelos de escenarios extraídos del juego *Panzer General*©, es decir, serán capturas de pantalla de áreas concretas de dichos escenarios, bajo los que se ha definido la capa de información subyacente que modelará dicha zona. Las dimensiones de cada una de estas áreas corresponderán a una zona (casi) cuadrada de unos 22,5 Km de lado (considerando una longitud de celda de unos 500 m, como se indica en la Figura 2.1), es decir, unos 500 Km^2.

7.1.1. Mapa Panzer General - Dos Enemigos y Ríos

Como se puede ver en la Figura 7.1 se trata de un escenario de 45x45 celdas que representa una llanura con varios núcleos población y muchas zonas boscosas, aunque dispersas. En la parte oeste del mismo, se pueden ver varias elevaciones montañosas y zonas de bosque, mientras que en la zona sur-suroeste, se tienen varios ríos conectados entre si y cruzados por algunos puentes. Existen dos enemigos en el mapa (señalados con sendas celdas con borde rojo), ambos situados en zonas más altas para tener un mayor campo visual, y encontrándose uno de ellos simplemente vigilando y otro armado y preparado para batir una zona a su alrededor y los puentes más cercanos al mismo (con armas de largo alcance). Las zonas afectadas por las armas están marcadas con celdas en color rojo (en el mapa de la derecha), siendo el rojo más intenso cuanto mayor letalidad tenga asociada cada una de ellas. La unidad del problema está situada en la zona sur del mapa (con borde negro), mientras que el punto de destino se encuentra señalado al norte del mismo (con borde amarillo).

En la Figura 7.2 se pueden ver las zonas definidas por la capacidad de adquisición, tanto de la CIA (área con trama amarilla), como de cada uno

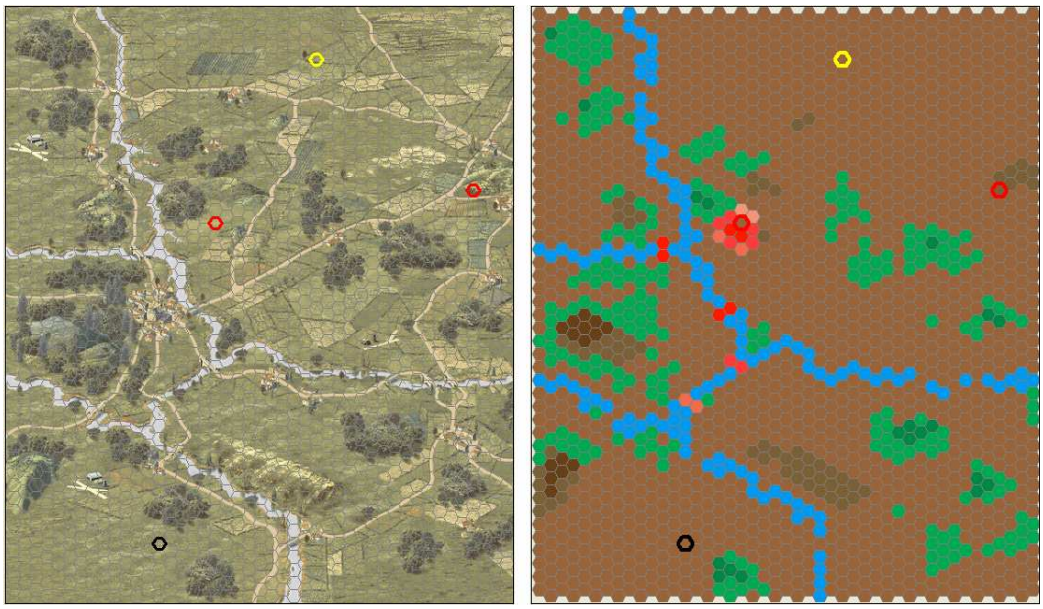

Figura 7.1: Ejemplo de modelado de un mapa real en el que se tienen un lago, ríos y mucha vegetación. A la derecha el mapa real y a la izquierda la capa de información inferior.

de los enemigos (áreas con trama roja). En ambos casos, se ha considerado una distancia de unos 9 Km (fiel a la realidad), por lo que en base a las dimensiones que se presentaron en la Figura 2.1, se corresponde con el área definida considerando 18 celdas de radio (hexagonal por el tipo de grid en el que se modela el escenario), ya que cada una tiene una longitud de unos 500 m. En la misma figura, pero a la derecha, se muestran las celdas vistas y ocultas desde el punto de vista de los enemigos. Las celdas sombreadas (con trama negra) son ocultas al mismo. Para determinar la ocultación de cada una de ellas, se recurre a la función que determina la *línea de visión* entre un enemigo y cada celda (limitada a su vez por la mencionada capacidad de adquisición), la cual está comentada en el Apéndice B5. En este caso no hay obstáculos naturales que señalar en el escenario.

Según los miembros del MADOC, la dificultad de resolución de este escenario sería de 3, en una escala de 1 a 4, siendo 4 la máxima dificultad. Este nivel de dificultad es debido a que existen dos enemigos a considerar y además uno de ellos armado. También hay que considerar que entre ambos

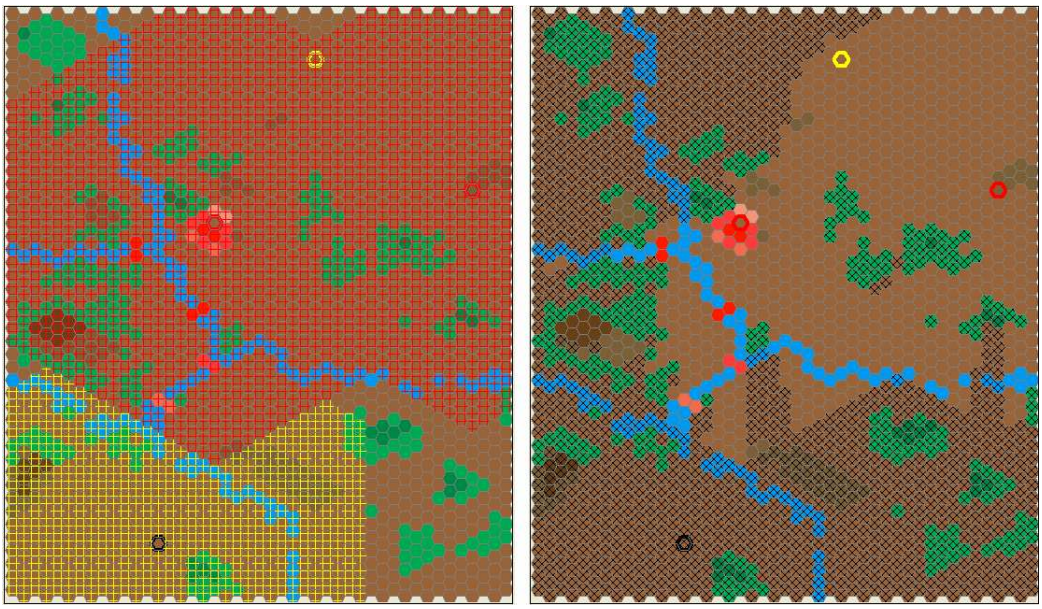

Figura 7.2: Mapa real modelado en la Figura 7.1, en el que se han señalado a la derecha las áreas correspondientes a la capacidad de adquisición de los enemigos (celdas con tramas en rojo) y de la unidad (celdas con tramas en amarillo). A la izquierda, se han marcado las zonas vistas y ocultas desde el punto de vista de los enemigos, siendo ocultas las celdas sombreadas.

vigilan casi todo el mapa (ver Figura 7.2), quedando pocos resquicios seguros para la unidad.

Se puede notar a simple vista, que las soluciones que tiendan hacia la seguridad deberían discurrir por celdas fuera del campo de visión de las unidades enemigas, por tanto, se moverían por celdas fuera de las áreas que determinan su respectivas capacidades de adquisición, o bien por las celdas tras otras que obstruyan la línea de visión de los mismos, como celdas entre montañas o bosques. Los caminos con tendencia a la rapidez, serán muy directos, pero su coste en seguridad (salud) será enorme, pues con toda certeza discurrirán en su mayor parte por zonas vistas por los enemigos e incluso batidas por las armas de uno de ellos.

7.1.2. Mapa Panzer General - Montañas

Este escenario se puede ver en la Figura 7.3. Al igual que en el caso anterior, se trata de un escenario de 45x45 celdas, representando en esta ocasión un área montañosa casi yerma, con algunas depresiones y varios picos, atravesada en su parte inferior por un río y con zonas de vegetación al norte.

En este caso, la unidad del problema está situada en la zona suroeste del mapa (celda con borde negro), y el punto de destino está marcado tras la montaña más grande al norte (celda con borde amarillo). En este escenario no hay unidad enemiga conocida.

Figura 7.3: Mapa de 45x45 en el que se muestra una zona montañosa con varias depresiones y picos. La unidad está situada al suroeste (con borde negro) y el punto de destino al norte (con borde amarillo). La imagen de la derecha corresponde a la capa de información subyacente que modela el mapa realista de la izquierda.

La Figura 7.4 muestra datos de interés del escenario. A la izquierda se pueden ver los obstáculos naturales que se pueden encontrar en el mapa. Éstos son, fronteras entre celdas que la unidad no puede atravesar por tener una

diferencia de altura mayor del número de niveles para los que está capacitada (en este caso 2 niveles). Están marcados con gruesas líneas negras.

Como ya se ha comentado, en este escenario no hay enemigos, por lo que no habrá celdas vistas y ocultas que considerar a priori. Aunque durante la búsqueda de la solución si que se hace un estudio de la ocultación de las celdas dentro de una zona, la cual está definida por la capacidad de adquisición de la CIA. Se trata del caso en el que no hay enemigos conocidos que se comentó en la definición de la función O en las funciones heurísticas comentadas en la Sección 5.2.1.1. Es decir, antes de buscar el camino solución, a cada celda del mapa se le asocia un *valor de ocultación* en base al número de celdas a las que sea oculta (en media) dentro de un área a su alrededor, determinada por la capacidad de adquisición de la unidad, como ya se ha dicho. Por ello, en la Figura 7.4, a la derecha, se puede ver el ejemplo de la zona a valorar para la celda marcada con un punto negro (situada en el centro), considerando como capacidad de adquisición 18 celdas, como ya se comentó anteriormente. Si bien se podrían considerar áreas menores si así se desea (por cuestiones de mejora de tiempos de ejecución, por ejemplo).

La dificultad asociada a este mapa es en este caso 2, porque aunque no hay enemigos en el mismo a priori, la unidad debería transitar por zonas lo más ocultas posible para evitar un posible ataque sorpresa o emboscada (consideración importante dentro de la táctica militar). Por tanto, cuando se busquen caminos tendentes a la rapidez, la unidad se moverá de forma muy directa entre origen y destino, únicamente dando un pequeño rodeo en caso de encontrar obstáculos naturales. Por otra parte, si lo que se pretende encontrar son los caminos más seguros, la ruta será similar (bastante dirigida hacia el destino) ya que no hay enemigo del que preocuparse, si bien, se tendrá más en cuenta la ocultación de las celdas cercanas a dicho camino respecto de las de su entorno, para intentar evitar en la medida de lo posible futuros ataques.

La resolución de estos escenarios se mostrará posteriormente en la Sección 7.4, si bien antes, en los apartados a continuación del actual, se expondrán los parámetros a utilizar en los experimentos, así como sus valores y justificación. Del mismo modo, se presentarán varios ajustes de parámetros con el fin de conseguir soluciones interesantes para la comparativa entre los diferentes algoritmos.

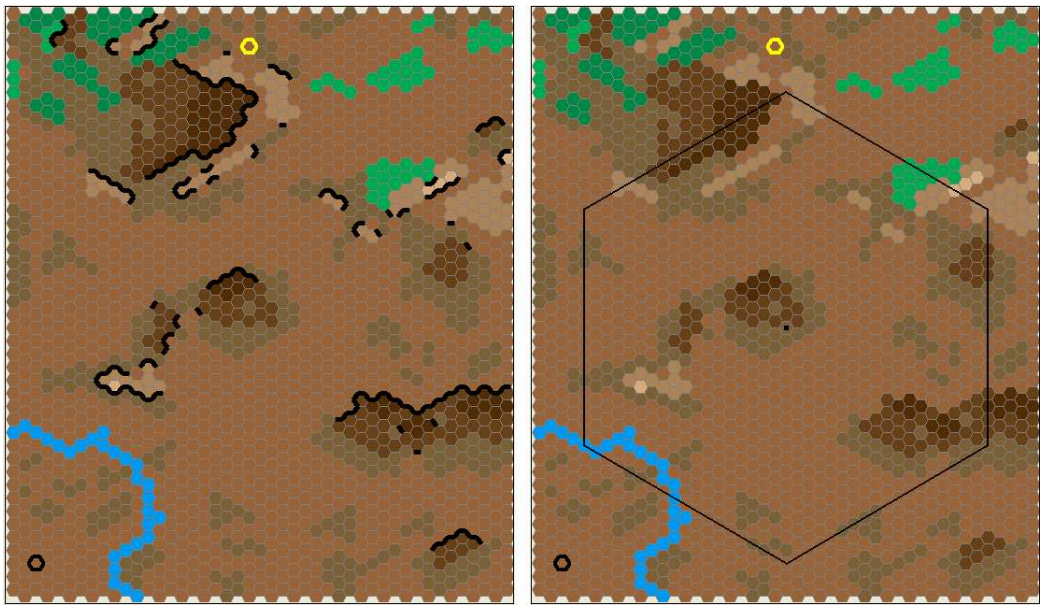

Figura 7.4: Mapa real modelado en la Figura 7.3, en el que se han señalado a la izquierda, los obstáculos naturales que la unidad no puede atravesar (diferencias de altura mayor de 2 entre celdas vecinas). A la derecha se muestra la zona a valorar cuando se considera la celda marcada con un punto negro (área definida por la capacidad de adquisición).

7.2. Parámetros para la experimentación

En esta sección se presentarán los diferentes valores para los parámetros y pesos a usar en los algoritmos comentados anteriormente, así como los parámetros/restricciones a tener en cuenta para la resolución de los escenarios planteados en la sección anterior.

En primer lugar, cabe comentar que los valores de los parámetros y pesos mostrados en las fórmulas de las descripciones de los algoritmos (Capítulos 5 y 6) afectan en gran medida al funcionamiento de cada uno de ellos. De tal forma que con los parámetros se establece el equilibrio entre la exploración y la explotación más acorde al problema a resolver, y además se dirige la búsqueda en función de la dificultad y propiedades de dicho problema. Por su parte, los pesos fijan la importancia de los miembros de las heurísticas y las funciones de evaluación, a fin de decantar la búsqueda de soluciones hacia

las zonas más prometedoras de acuerdo al criterio deseado, y valorarlas según dicho criterio respectivamente.

Ambos grupos han sido estudiados mediante experimentación sistemática y refinados hasta conseguir un conjunto que consiga que los algoritmos se comporten de la forma deseada en todo tipo de escenarios.

Las restricciones determinan ciertas propiedades de los problemas (escenarios) a resolver, las cuales limitan el conjunto de soluciones posibles.

7.2.1. Parámetros de los algoritmos

Los parámetros a considerar por los algoritmos han sido introducidos previamente en los capítulos en los que éstos se describieron (5 y 6). Muchos de esos parámetros son comunes a todos los algoritmos, mientras que otros solo los utilizan algunos. A continuación se describen dichos los parámetros y se señalan los algoritmos que los consideran:

- λ ($\in [0, 1]$): sin duda se trata del parámetro más importante en la aplicación de los algoritmos de este trabajo. Como ya se comentó en la presentación de cada algoritmo que lo considera, con él se establece la prioridad de cada uno de los objetivos principales (rapidez y seguridad) en la búsqueda. Es utilizado por los algoritmos:

  - CHAC: se trata de un parámetro definido a priori (antes de la ejecución), para determinar la importancia relativa de cada uno de los dos objetivos. De forma que si toma un valor cercano a 1, tendrá mayor prioridad la búsqueda de *caminos rápidos*, y en caso de ser cercano a 0, la prioridad la tendrá la búsqueda de *caminos seguros*. Únicamente lo usa la implementación que aplica la RTC.
  - CHAC-4: éste utiliza dicho parámetro siguiendo el mismo esquema que CHAC, pero asociándolo a la prioridad de los subobjetivos dos a dos. Esto es, si λ tiene un valor cercano a 1, la mayor prioridad la tendrán los subobjetivos relacionados con la rapidez, es decir, el *consumo de recursos* y la *distancia media al destino*. Si por el contrario, dicho valor es cercano a 0, los objetivos prioritarios serán los relacionados con la seguridad, por tanto el *consumo de recursos* y la *ocultación*. Al igual que en el caso de CHAC, únicamente es utilizado por la implementación que considera la RTC.

  - MOACS: de nuevo el parámetro toma un valor fijo durante toda la ejecución, teniendo el mismo significado y relevancia que en el caso de CHAC.
  - BiAnt: al igual que en los casos anteriores, también se considera su valor como fijo durante toda la ejecución.

- q_0 ($\in [0, 1]$): el segundo en importancia. Se trata de un parámetro estándar de los SCHs (ver Sección 3.4.2), mediante el cual se define la regla pseudoaleatoria de transición de estados (como se puede ver en la misma sección). Su influencia determina en gran medida la búsqueda, pues con él, es posible establecer la relación entre exploración (búsqueda de nuevas soluciones) y explotación (mejora de las soluciones conocidas), según las características del algoritmo y del problema a resolver. De modo que si su valor es cercano a 1, la búsqueda se decanta hacia la explotación de las soluciones conocidas. En caso contrario (si es cercano a 0), la búsqueda tendrá un factor exploratorio más alto, por lo que se tenderá a buscar nuevas soluciones (trabajando en nuevas zonas del espacio de búsqueda). Al ser un parámetro propio de los SCHs, lo utilizarán CHAC, mono-CHAC, CHAC-4 y MOACS.

- α: es el *factor de ponderación de feromona* utilizado en todos los algoritmos de OCH (como se puede ver en la Sección 3.4). Fija la importancia de la información memorística respecto a la heurística en la búsqueda, sobretodo en la regla de transición. Si su valor es grande, las concentraciones de feromona tendrán mucha relevancia a la hora de seleccionar el siguiente nodo en la construcción de una solución. Si dicho valor llegara a ser 0, no se tendría en cuenta la información de los rastros de feromona en la búsqueda, lo que significaría que ésta estaría guiada únicamente por heurísticas, convirtiendo el algoritmo en una búsqueda voraz. Dado que se trata de un parámetro propio de los algoritmos de OCH, lo considerarán CHAC, mono-CHAC, CHAC-4, MOACS y BiAnt.

- β: es el *factor de ponderación de heurística*, también utilizado en la inmensa mayoría de algoritmos de OCH (ver también la Sección 3.4). En este caso, determina la importancia de la información heurística respecto a la feromona en la búsqueda, siendo usando nuevamente en la regla de transición. Valores grandes para este parámetro implicarán una

gran relevancia de esta información en la elección del siguiente nodo, seleccionando con mayor probabilidad aquel que mayor valor heurístico tenga asociado. Si tomase un valor igual a 0, no se consideraría ningún conocimiento heurístico en la búsqueda, siguiendo únicamente los rastros existentes, lo que llevaría a un estancamiento del algoritmo al considerar siempre los mismos nodos (pertenecientes a las soluciones obtenidas previamente por otras hormigas). Este parámetro será utilizado por los mismos algoritmos que el anterior.

- ρ ($\in (0, 1)$): otro parámetro clásico de los algoritmos de OCH. Se trata del llamado *factor de evaporación de feromona*, el cual determina la tasa en la que se reduce la cantidad de feromona en cada uno de los rastros a evaporar (la elección de los cuales depende del algoritmo). De modo que si toma un valor cercano a 1, dicha evaporación será muy brusca, haciendo que las hormigas 'olviden' rápidamente lo aprendido, lo cual conllevará la búsqueda de soluciones en zonas aún no exploradas. Es decir, este parámetro también influye en el aumento de la exploración durante la ejecución del algoritmo. Nuevamente se puede consultar su forma de uso en la Sección 3.4. También será considerado por todos los algoritmos excepto GRMO.

Los valores considerados para cada uno de estos parámetros se pueden ver en la Tabla 7.1.

<table>
<tr><th></th><th>mono-CHAC</th><th>CHAC</th><th>CHAC-4</th><th>MOACS</th><th>BiAnt</th><th>GRMO</th></tr>
<tr><td>λ</td><td>-</td><td colspan="4">0,9 — 0,1</td><td>-</td></tr>
<tr><td>q_0</td><td colspan="4">0,4</td><td>-</td><td>-</td></tr>
<tr><td>α</td><td colspan="5">1</td><td>-</td></tr>
<tr><td>β</td><td colspan="5">2</td><td>-</td></tr>
<tr><td>ρ</td><td colspan="5">0,1</td><td>-</td></tr>
</table>

Tabla 7.1: Valores de los parámetros utilizados para cada uno de los algoritmos en los experimentos.

Los valores mostrados en dicha tabla para α, β y ρ son los habituales en los algoritmos de OCH, es decir, los que se ha demostrado que dan mejores resultados en la gran mayoría de los casos. Para justificar dichos valores existen varios estudios, desde el inicial de Dorigo et al. [15] hasta otros más recientes [99, 100].

El valor para q_0 ha sido fijado tras un proceso de experimentación heurística como el que mejor se ajusta a todo tipo de escenarios. Este parámetro, como se comentó previamente, tiene influencia directa en la forma en que se lleva a cabo la búsqueda (construcción) de soluciones, pues de él depende que se exploten soluciones conocidas o se explore para obtener nuevas. Esta condición, en el problema que se resuelve en este trabajo, depende en gran medida del escenario a resolver, pues cada uno tendrá una dificultad asociada que implicará la necesidad de un mayor factor de exploración (si hay pocos caminos buenos) o de explotación (si hay muchos caminos posibles). Generalmente, los escenarios contienen múltiples caminos solución y será más recomendable 'refinar' los conocidos que buscar continuamente nuevos, pero peores. Por eso, el valor que se ha asignado a q_0 hace que en muchas ocasiones (casi en la mitad de las elecciones), una hormiga se decante por seguir el nodo más recomendable según la heurística y los rastros de feromona precedentes, lo cual se traduce en que dicha hormiga construye una solución 'parecida' a las de las demás, pero que puede resultar mejor globalmente.

Respecto a los valores a considerar para el parámetro λ, como se puede ver, son dos distintos, empleándose respectivamente para buscar los *caminos más rápidos* y los *caminos más seguros*. Si bien dado que no toman los valores extremos (1 y 0), deberían considerarse como *caminos muy rápidos* y *caminos muy seguros*, pero que se denominarán de esa forma porque desde el punto de vista militar, se considera que como mínimo debería tenerse en cuenta siempre cada uno de los objetivos en un 10 % respecto al otro, es decir, para buscar un camino lo más rápido posible, no es de recibo obviar completamente la seguridad, pues de lo contrario podrían mermarse los efectivos de la unidad (salud) en exceso. Del mismo modo, en la búsqueda del camino más seguro, no deben desecharse por completo las consideraciones relativas a la rapidez, pues éstas aseguran tanto que el camino se dirija hacia el destino deseado, como que la unidad disponga de recursos suficientes para alcanzar dicho destino. En cualquier caso, también se harán experimentos utilizando los valores extremos de λ (1 y 0), como se verá en la Sección 7.3, los cuales se complementarán con configuraciones especiales de los pesos que se utilizan en la funciones heurísticas de los algoritmos para obtener resultados 'extremos' (*caminos extremadamente rápidos* y *caminos extremadamente seguros*) que sirvan como base comparativa con los demás resultados.

Por último, se puede ver que el algoritmo GRMO no hace uso de ninguno de estos parámetros. En cambio todos los demás algoritmos utilizarán los mismos valores para poder comparar sus resultados en las mismas condi-

ciones.

7.2.2. Pesos

En varias de las ecuaciones que definen tanto los algoritmos propuestos, como los adaptados (Capítulos 5 y 6), existen algunas funciones formadas como agregación varios términos. Es decir, se pretende modelar una única función que depende a su vez de cada uno de esos términos. Al mismo tiempo, cada uno de ellos tendrá normalmente una importancia asociada dependiente del significado (o valor) que se quiera dar a la función global, de modo que para asignar esa importancia, cada término x dentro de la función f es ponderado utilizando un *peso*, notado generalmente como ω_f^x.

Como se puede ver en dichos capítulos, los pesos han sido utilizados en las definiciones de las *funciones heurísticas*, así como en las ecuaciones que definen las *funciones de evaluación*. Dichas funciones se definieron como agregativas inicialmente en el algoritmo CHAC (Secciones 5.1.1.1 y 5.1.1.3) y posteriormente fueron adaptadas al algoritmo mono-CHAC. Además, los algoritmos MOACS, BiAnt y GRMO utilizan las mismas funciones, por lo que todos ellos hacen uso de los mismos pesos para ponderar los términos. Únicamente CHAC-4 carece de pesos y ponderaciones, ya que la idea de dicho algoritmo es tratar cada término como un objetivo por separado.

Por tanto, se tienen pesos definidos dentro de dos pares de funciones (dos funciones heurísticas y dos funciones de evaluación), y existe una función de cada tipo por cada objetivo. De modo que en cada función, los pesos deberán ajustarse para asignar la importancia relativa de cada uno de los términos respecto del objetivo al que se refiera dicha función. Es decir, en la función heurística creada para que las soluciones satisfagan el objetivo de la *rapidez*, deberían tener un peso mayor los términos relacionados con dicho objetivo, como serían el bajo consumo de recursos (ω_r^r) y la reducción de la distancia al objetivo (ω_r^d). Por contra, en la función heurística dedicada al objetivo de la *seguridad*, la mayor ponderación debería recaer sobre los términos relacionados con ella, es decir, el bajo consumo de salud (ω_s^s) y la ocultación de las celdas del camino (ω_s^o). Si bien, cada una de esas funciones incorpora además un término centrado en el otro objetivo para tenerlo siempre en cuenta en la búsqueda, estos son la distancia al destino (ω_s^d) en la heurística de la seguridad (para que el camino sea dirigido) y la ocultación (ω_r^o) en la heurística de la rapidez (para moverse por celdas relativamente seguras). Dichos términos, por tanto tendrán asignado un peso mucho menor que los demás.

Respecto a las funciones de evaluación, ambas (tanto la referida a un objetivo, como la otra) tienen un término relativo a la ocultación, pues es muy importante la previsión de ataques futuros (según las doctrinas de la táctica militar). Dicho término estará definido al margen del principal dentro de cada función, el cual será respectivamente: el consumo de recursos en la función de evaluación de caminos rápidos, y el consumo de salud en la función que evalúa los caminos seguros. Por consiguiente, en el primer caso el término relativo a la ocultación tendrá muy poca relevancia, mientras que en el segundo tendrá incluso más relevancia que el llamado término principal.

En la Tabla 7.2, se pueden ver los valores asignados a cada uno de los citados pesos.

Objetivo Rapidez	*Objetivo Seguridad*
$\omega_r^r = 0{,}15$	$\omega_s^s = 0{,}15$
$\omega_r^d = 0{,}80$	$\omega_s^d = 0{,}05$
$\omega_r^o = 0{,}05$	$\omega_s^o = 0{,}80$
$\omega_{F,r}^o = 0{,}5$	$\omega_{F,s}^o = 10$

Tabla 7.2: Pesos asignados a los términos dentro de las funciones relativas a cada objetivo. Arriba los pesos de las funciones heurísticas y abajo los de las funciones de evaluación.

Esos valores han sido fijados usando experimentación sistemática, siendo los que mejores resultados arrojan en la gran mayoría de escenarios.

Como se puede ver, los valores de los pesos usados en las funciones heurísticas están normalizados. Dichos valores son bastante lógicos, pues la reducción de la distancia al destino (camino lo más directo posible) es el término más importante en los caminos rápidos, seguido del consumo de recursos (celdas a atravesar con menos coste en recursos) y con muy poca relevancia la ocultación. En el caso de los caminos seguros, la ocultación es la que cobra mayor importancia (para evitar ataques futuros), seguida del consumo de salud (celdas sin letalidad y con bajo coste en salud), y por último la reducción de la distancia al destino, sin mucha trascendencia.

Los pesos relativos a las funciones de evaluación ponderan de manera absoluta el término de la ocultación. Esto es así por dos motivos: por una parte, para normalizar los pesos habría que incluir otros dos que ponderasen a los miembros referidos al consumo de recursos y salud de cada una de las funciones de evaluación (ver Ecuaciones 5.11 y 5.12), lo cual se estimó que

sería contraproducente. Por otra parte, los valores asociados tienen un orden de magnitud adecuado para la influencia que se quiere provocar por parte de la función de evaluación en la búsqueda (en el aporte global de feromona). De ahí que el peso de la ocultación en el caso de la búsqueda de caminos rápidos sea muy pequeño y en el caso de caminos seguros tome un valor bastante grande.

7.2.3. Restricciones o características del problema

Las *restricciones* son parámetros que tipifican el problema, es decir, añaden ciertas propiedades al mismo en general o a alguno de sus elementos de forma que se limita el conjunto de soluciones posible o la forma de encontrarlas.

Dado que los problemas que se atacan en este estudio son escenarios que modelan campos de batalla militares, estos parámetros estarán estrechamente relacionados con el entorno o los 'actores' del mismo, es decir, serán propiedades tanto de los mapas en si, como de las unidades que haya en el mismo. De forma que las primeras están directamente relacionadas con el posible espacio de búsqueda de soluciones, pues refuerzan o relajan restricciones que hacen factibles unas soluciones u otras. Dichas restricciones las impone en este caso la definición del escenario, en el que la situación de la unidad o de los enemigos por ejemplo, cambia completamente el conjunto de soluciones posibles. Otro ejemplo sería la localización de obstáculos artificiales, que la CIA no podría atravesar.

Las propiedades relativas a los 'actores' del problema limitarán generalmente la forma de encontrar las soluciones, sobretodo aquellas que se refieran a la unidad que debe recorrer los caminos solución, si bien muchas de sus características, actuarán como restricciones dentro del escenario, como se verá más a continuación.

Las restricciones/características de los problemas a considerar son:

- *Nivel/Puntos de recursos de la CIA*: se trata de un número que indica el máximo nivel de recursos (modelando como se puede ver en la Tabla 2.1 el nivel de combustible, víveres, etc) de que dispone la unidad para recorrer cada uno de los posibles caminos solución. Este número restringe la cantidad y tipo de celdas por las que discurrirá dicho camino. De modo que mientras se construye una solución, se va consumiendo el nivel de recursos, por lo que podría darse el caso de no disponer de suficientes recursos restantes para pasar a un determinado nodo. Es decir,

dicha propiedad limitará el conjunto de soluciones posibles (siempre que tenga un valor ajustado al escenario/problema a resolver) y a su vez, condiciona la forma de búsqueda de dichas soluciones.

- *Nivel/Puntos de salud de la CIA*: número que indica el máximo número de puntos de salud de los que dispone la unidad. Modela como se puede observar en la Tabla 2.1, la salud global de los soldados y de los vehículos de la CIA. Este número limita el número de celdas dañinas a atravesar por la unidad, puesto que al igual que en el caso anterior, dicha salud se va consumiendo en cada celda que se añade al camino (mucho más si ésta tiene una letalidad asociada). Nuevamente limita el conjunto de soluciones posible (si tiene un valor ajustado) y condiciona la búsqueda.

- *Capacidad de adquisición de la CIA*: máxima distancia a la que puede 'mirar' dicha unidad. Esta propiedad influye en gran medida en la forma en que se realiza la búsqueda en un mapa, sobretodo cuando no existen enemigos conocidos, ya que limita la zona a explorar en el escenario, antes de decidir la siguiente celda del camino.

- *Capacidad de adquisición de los enemigos*: máxima distancia a la que alcanzaría la línea de visión de cada enemigo. Esto cambia el conjunto de soluciones a encontrar, pues intrínsecamente cambia el mapa, dado que la ocultación de las celdas del mismo depende en gran medida de este límite.

- *Máxima diferencia de altura que puede atravesar la CIA*: número que indica la diferencia de altura entre dos celdas vecinas que puede atravesar la unidad, cualquier diferencia mayor será considerada un obstáculo natural, el cual no podrá ser atravesado por la CIA. Nuevamente se restringe el conjunto de soluciones posible y se afecta de forma implícita al escenario, ya que este límite define zonas de obstáculos insalvables para la unidad en las zonas del mapa en las que se supere la diferencia de altura máxima que ésta puede atravesar.

Estos parámetros y las funciones que los consideran, están comentadas en el Apéndice B (y como condiciones/restricciones de simulador en B5).

Los valores considerados para cada una de estas restricciones/características están reflejados en la Tabla 7.3.

	mono-CHAC	*CHAC*	*CHAC-4*	*MOACS*	*BiAnt*	*GRMO*
Recursos	1000 puntos					
Salud	1000 puntos					
Capac_Adq_CIA	18 celdas					
Capac_Adq_ENE	18 celdas					
Max_Alt_Atrav	2—1					

Tabla 7.3: Valores de los parámetros/restricciones utilizados para cada uno de los algoritmos en los experimentos.

Como se puede ver, dichos valores son comunes a todos los algoritmos, pues lo que se pretende es compararlos en las mismas condiciones. Cabe destacar que se han dispuesto puntos tanto de recursos, como de salud suficientes para recorrer cualquier camino en los escenarios con los que se experimentará, pues el cometido de este estudio es comparar el rendimiento de los algoritmos y la calidad de las soluciones obtenidas, sin tener que interponerles trabas que limiten en demasía las posibles soluciones a obtener.

Respecto a los valores considerados para la capacidad de adquisición, tanto de la unidad, como de los enemigos, se han establecido 18 celdas como distancia límite, por ser equivalente a la distancia real a la que pueden 'ver' estas unidades, según se comentó previamente.

Por último, respecto a la máxima diferencia de altura que puede atravesar la unidad, se ha considerado un valor de 2 en general, si bien, en uno de los escenarios se establecerá éste como 1, para marcar más obstáculos naturales y ver cómo se comportan los algoritmos ante restricciones que limiten el discurrir de los caminos solución.

7.3. Variaciones de los algoritmos en base a la parametrización

En este apartado se comentarán algunas variantes que se han desarrollado a partir de los algoritmos comentados en los apartados anteriores, en concreto a partir de CHAC (Sección 5.1) y a partir de CHAC-4 (Sección 5.3), por ser dos representantes de los dos enfoques multiobjetivo que se tienen en este trabajo (el primero trabaja con dos objetivos y el segundo con cuatro).

Dichas variantes más que nuevas implementaciones son más bien *configuraciones* de parámetros y pesos, aplicadas sobre los dos algoritmos multi-

objetivo comentados, que pretenden obtener resultados para fijar unas bases comparativas a tener en cuenta en los experimentos.

7.3.1. CHAC-extremos

La idea de los algoritmos extremos es fijar, tanto el parámetro de control de prioridad, λ, como los pesos de las funciones heurísticas ($\omega^x_{r|s}$), para que la búsqueda se decante completamente por minimizar uno de los objetivos (sin considerar para nada el otro).

Los valores asignados a los diferentes parámetros y pesos en cada uno de los enfoques se resumen en la Tabla 7.4.

	Extremadamente Rápido	*Extremadamente Seguro*
λ	1	0
ω^r_r	0,15	-
ω^d_r	0,85	-
ω^o_r	0	-
ω^s_s	-	0,15
ω^d_s	-	0,01
ω^o_s	-	0,84

Tabla 7.4: Configuración de parámetros para algoritmo CHAC extremo

Dichos valores han sido obtenidos mediante experimentación heurística y están todos normalizados.

Como se puede observar en la Tabla 7.4, el parámetro principal para fijar la importancia de los objetivos es λ. En un enfoque extremo, éste tomará su valor más grande o más pequeño, dependiendo del objetivo que se pretenda considerar como único.

De modo que, en el caso del algoritmo *extremadamente rápido*, su valor será igual a 1, lo que llevará a eliminar toda repercusión de heurística y feromona del objetivo del camino seguro (η_s y τ_s, respectivamente), como se puede ver en la regla de transición (Ecuaciones 5.3 y 5.4). Es por eso que los pesos, tanto de la función heurística para camino seguro, como de la función de evaluación para camino seguro, no tienen repercusión alguna (no importa su valor).

Se podría pensar que la función de evaluación si que podría ser relevante, ya que de su valor depende la actualización global de feromona, pero nueva-

mente su trascendencia es nula durante la búsqueda del camino que satisfaga únicamente el otro objetivo, pues como ya se ha comentado, la feromona relativa a dicho objetivo no es considerada en la regla de transición que dirige a las hormigas.

Estas conclusiones son igualmente válidas a la inversa (con $\lambda = 0$), de ahí que los pesos relativos a la búsqueda de caminos rápidos, cuando prevalece únicamente el objetivo de la seguridad, no tengan relevancia alguna.

Por otra parte, viendo los pesos que si que tienen repercusión y que aparecen con un valor asociado en la Tabla 7.4, se puede notar que, en el caso de la búsqueda del camino *extremadamente rápido*, el peso relativo al consumo de recursos es importante (un 15 % del total), pero el peso de la minimización de la distancia tiene una importancia muy superior (un 85 %), puesto que, un camino directo al destino es sinónimo de un camino rápido y, por ende, un camino que requiere pocos recursos (considerando los consumos máximos en recursos que supone atravesar una celda cualquiera). Si la importancia fuese al contrario, las hormigas se moverían buscando celdas que consumiesen pocos recursos, pero los caminos podrían ser más largos, lo que implicaría un mayor coste total de recursos. Como es lógico, el parámetro que pondera la ocultación de las celdas toma un valor igual a 0, pues ésta está más relacionada con la seguridad que con la rapidez.

Mirando nuevamente la tabla, pero ahora considerando los pesos relativos a la seguridad en el caso de la búsqueda del camino *extremadamente seguro*, se podrá ver que son similares a los ya comentados, de modo que el peso relativo al consumo de salud que implican las celdas toma nuevamente un valor de importancia media (15 % del total), y el peso relativo a la ocultación una importancia grande (84 % del total), puesto que la ocultación, como ya se ha comentado anteriormente está estrechamente relacionada con el evitar futuros enfrentamientos con el enemigo que puedan repercutir en una gran cantidad de bajas (si se produce por sorpresa). En este caso, si que se asigna una importancia (aunque mínima) al peso referido a la reducción de distancia en los caminos seguros (un 1 % del total), puesto que sin este factor y sin rastro de la heurística que guía a las hormigas hacia el destino en los caminos rápidos (por ser $\lambda = 0$), las hormigas tendrían difícil llegar al nodo final deseado, lo cual no sería de recibo en un algoritmo de búsqueda de camino óptimo entre dos puntos.

7.3.2. CHAC-4-extremos

Del mismo modo que en el caso del algoritmo bi-criterio CHAC, también se han creado dos versiones *extremas* del algoritmo CHAC-4. La filosofía de dichas versiones es la misma que la de las comentadas anteriormente, de forma que la búsqueda se centrará en uno de los objetivos desechando completamente al otro. Esto se hace considerando los objetivos principales, es decir, rapidez y seguridad, puesto que es sobre éstos sobre los que se desea aplicar la ponderación que nos ofrece el parámetro λ. Por tanto, mediante dicho parámetro simplemente serán relevantes dos de los subobjetivos (y se desecharán los otros dos), respectivamente:

- *consumo de recursos y distancia media al objetivo*: si se prefiere considerar únicamente el objetivo de la rapidez ($\lambda = 1$).

- *consumo de salud y visibilidad*: si se desea tener en cuenta solamente el objetivo de la seguridad ($\lambda = 0$)[1].

En cualquier caso la búsqueda de caminos *extremadamente rápidos* o *extremadamente seguros* sólo dependerá de dicho parámetro, y no será necesario cambiar el valor de ningún otro. Se debe tener en cuenta que en la implementación del algoritmo CHAC-4 se eliminaron todos los pesos que sí se tenían en las funciones heurísticas y de evaluación del algoritmo CHAC (y que se han visto cambiar en la sección anterior). Éstos no son necesarios puesto que en este algoritmo, cada uno de los miembros que se ponderaban en CHAC es un objetivo independiente y no es combinado con los demás para obtener un valor heurístico, ni para evaluar el camino solución.

7.4. Resolución de escenarios

En esta sección se resolverá cada uno de los escenarios descritos en la primera sección del presente capítulo (7.1), haciendo uso de los algoritmos presentados en los Capítulos 5 y 6, considerando los primeros como originales de este estudio, y los demás como algoritmos introducidos previamente en la bibliografía y adaptados a la resolución del problema de camino óptimo

[1]En realidad $\lambda = 0{,}001$, ya que al considerarse objetivos separados, con 0 se desecharía completamente la heurística de la reducción de distancia, que es la que dirige el movimiento hacia el destino.

multiobjetivo (PCOMO) en escenarios militares. Si bien, se podría considerar que la implementación del algoritmo greedy multiobjetivo (GRMO), también es novedosa.

Una vez resuelto cada uno de los escenarios por todos los algoritmos, se presentarán los resultados de éstos de manera estructurada mediante tablas, a fin de poder analizar dichos datos y establecer comparativas entre los distintos métodos en base a la bondad de los mismos. A su vez, y dado que los problemas a resolver son escenarios modelados en un simulador (mSS), también se presentarán las mejores soluciones obtenidas por cada uno de los algoritmos de forma gráfica, señalando en el mapa resuelto, cada uno de los caminos solución elegidos. Estos resultados también serán sometidos a un análisis y una comparativa, pues complementan la información de las tablas y resultan más comprensibles y explícitos en muchos casos.

7.4.1. Consideraciones previas a la resolución de escenarios

Existen varios aspectos que deben ser comentados antes de realizar los experimentos sobre los problemas planteados.

El primero y más importante se refiere a los conjuntos de soluciones obtenidas por cada uno de los algoritmos. Dado que todos ellos excepto mono-CHAC, son algoritmos multiobjetivo (MO), en cada ejecución se obtiene un conjunto de soluciones no dominadas conocido como Conjunto o Frente de Pareto (FP) (ver sección 4.1). Normalmente, el FP obtenido por un algoritmo MO debe ser lo más grande y variado posible (con mucha diversidad a ser posible), estableciéndose incluso comparativas entre el FP obtenido y un frente ideal, conocido de antemano en muchos casos, para valorar la bondad de un nuevo algoritmo MO.

En el caso del problema que nos ocupa, se suceden varias tesituras que cambian esta situación y hacen que se obtenga *un FP no muy extenso*. La primera es que los PCO suelen tener un conjunto de soluciones bastante limitado (si se comparan con otro tipo de problemas como aproximación de funciones u optimización con variables reales), ya que entre dos puntos de un grafo finito el número de posibles rutas a trazar es pequeño y máxime si dichas rutas deben cumplir una serie de condiciones o restricciones, como en este caso. De modo que en cada escenario el conjunto de soluciones no dominadas no será muy alto. La segunda es que por la naturaleza del problema y el

entorno en el que se desarrolla, la variedad de soluciones no es un aspecto prioritario ni importante de los algoritmos estudiados en el presente trabajo[2]. Por este motivo, en todos los algoritmos, de todas las soluciones con los mismos costes únicamente se considerará una de ellas, desechándose el resto (aún pudiendo ser no dominadas), algo que normalmente no se haría en otro tipo de algoritmos MO. A esto se une el uso del parámetro λ en casi todos los algoritmos del estudio, el cual limita sobremanera la zona del espacio de búsqueda a considerar (según la prioridad que se asigne a cada objetivo), pues de todo el frente ideal de soluciones, centrará a los algoritmos en la zona en la que el objetivo prioritario tenga mejores resultados y, por tanto, limitará también la cantidad de soluciones que pueda encontrar cada uno de ellos.

Otro aspecto a tener en cuenta y que está estrechamente relacionado con el comentado acerca del FP y del número de soluciones que ofrecen los algoritmos es que *únicamente será relevante una solución por cada objetivo*, a efectos comparativos. Es decir, de todas las soluciones obtenidas en el FP final para un algoritmo en concreto, solamente se desea una (aunque todas son 'igualmente buenas') para establecer una comparación con las obtenidas por los demás algoritmos. Dicha solución se elegirá siguiendo el criterio militar[3] en el que si se da prioridad a un objetivo, éste debe prevalecer casi en cualquier caso; de modo que se la solución a considerar será aquella que mejor resultado haya obtenido para la función de evaluación referida al objetivo prioritario. Esto es, si un algoritmo realiza una búsqueda de soluciones en un escenario dando prioridad a la rapidez, del conjunto de soluciones no dominadas obtenidas al final (FP), se elegirá como 'la mejor' aquella que tenga el valor mínimo en la función de evaluación relativa a la rapidez (F_r). Este aspecto, si bien puede resultar chocante en un principio dado el tipo de algoritmos con los que se trabaja, tiene su justificación nuevamente en el entorno en el que se desarrollan dichos algoritmos, cuyo objetivo inicial era el de servir de apoyo a un usuario final que interactuase con los escenarios/problemas a través de un simulador. Por tanto, dicho usuario únicamente elegiría una solución entre todas las resultantes.

A fin de poder establecer una comparativa 'justa' entre todos los algoritmos, todos ellos han sido ejecutados en las mismas condiciones, es decir,

[2]Estos algoritmos fueron desarrollados para que los aplicase un usuario final por medio del simulador, y pudiese seleccionar las soluciones que más le conviniesen de forma sencilla

[3]según indicaron los miembros del MADOC

se han considerado los mismos parámetros y pesos (como se pudo ver en la Sección 7.2). Del mismo modo, todos se han ejecutado el mismo número de iteraciones y con el mismo número de hormigas (excepto GRMO que no trabaja con estos agentes). Este aspecto es también relevante, sobretodo desde el punto de vista de la parametrización, ya que por las propiedades de cada algoritmo, es muy posible que una configuración dedicada a cada uno de ellos hiciera que éstos obtuviesen mejores soluciones, pero esto dificultaría mucho la realización de una comparativa entre ellos.

Siguiendo la misma filosofía, todas las soluciones han sido evaluadas finalmente usando como funciones F_r y F_s, definidas en la Sección 5.1.1.3, es decir, los algoritmos bi-criterio por definición como CHAC, MOACS, BiAnt o GRMO, consideran esas funciones como funciones de evaluación durante su proceso. Pero otros algoritmos, como son mono-CHAC y CHAC-4, evalúan sus soluciones mediante las funciones definidas en las Secciones 5.2.1.3 y 5.3.1.3 respectivamente. Es decir, el primer algoritmo considera una única función de evaluación (F), mientras que el segundo utiliza cuatro funciones de evaluación (F_r, F_d, F_s y F_v), una por cada objetivo. Éstas funciones son utilizadas durante el desarrollo de cada uno de estos algoritmos, considerándose en las actualizaciones de feromona (si corresponde) y usándose como criterio de valoración de dominancia para la construcción del FP. Pero una vez concluida su ejecución y hallado el conjunto de soluciones resultante (las no dominadas dentro del FP), cada una de ellas es evaluada nuevamente considerando únicamente las funciones F_r y F_s. El motivo de esta evaluación es nuevamente tener la posibilidad de comparar las soluciones obtenidas con cada uno de los métodos, ya que los dos últimos trabajan con un número diferente de objetivos del resto, aunque todos buscan resolver el mismo problema en cada caso.

Para hacer la comparativa, se realizarán 30 ejecuciones de cada algoritmo, con cada regla de transición (para los que tengan varias) y sobre cada escenario, aplicando además cada vez una configuración para el parámetro λ para buscar soluciones a los caminos más rápidos y más seguros respectivamente. Se aplicarán 1500 iteraciones y se considerarán 50 hormigas en todos los casos (salvo GRMO que no utiliza hormigas). Se obtendrán 30 FP distintos (excepto mono-CHAC que sólo obtendrá 30 soluciones). De cada uno de esos frentes, se elegirá la solución que minimice el objetivo que tenga más prioridad en cada caso y se trabajará con esas 30 soluciones. De ellas se escogerá la mejor a su vez, y se calcularán las medias y desviaciones típicas para mostrarlas en tablas. Además, las mejores soluciones serán 'proyectadas'

sobre el escenario/problema que se esté resolviendo en cada caso mediante el simulador mSS y se mostrará una figura con dicha solución sobre el mapa.

No se han establecido comparativas entre los FPs puesto que el frente ideal no es conocido en estos problemas.

Todos los experimentos se han realizado en una única máquina, un portátil Pentium M 1.6GHz con 2GB de memoria RAM.

A continuación se describirán los experimentos llevados a cabo sobre cada escenario propuesto, realizando la comparativa entre las soluciones obtenidas por los algoritmos y analizando posteriormente dichas soluciones.

7.4.2. Mapa Panzer General - Dos Enemigos y Ríos

En primer lugar se ha afrontado la resolución del escenario definido en la Sección 7.1.1, al que nos referiremos en adelante como *MPG-2EneRios*. De modo que se han aplicado cada uno de los algoritmos comentados previamente, utilizando las diferentes reglas de transición (RTC y RTD) para los que tengan esa posibilidad, e incluyendo las variaciones extremas de CHAC y CHAC-4. Cada uno de estos métodos se ha aplicado a la búsqueda del camino más rápido y del más seguro. Se han llevado a cabo los experimentos, recopilado las soluciones, elegido las representativas de cada FP y calculado la media y la desviación típica de 30 ejecuciones en cada caso, de la forma que se ha indicado en la sección anterior.

Los resultados se encuentran resumidos en la Tabla 7.5.

Para poder interpretar la tabla, hay que notar que las soluciones (caminos) se han agrupado en dos columnas: la de la izquierda está dedicada a las soluciones obtenidas considerando como prioritario el objetivo de la rapidez, de ahí que esté marcada como *Más Rápido* o *Extremadamente Rápido* en el caso de la variaciones extremas (las cuales consideran los valores extremos de λ). La columna de la derecha por contra, está dedicada a las soluciones obtenidas considerando la seguridad como objetivo principal, por eso se han etiquetado como *Más Seguro* o *Extremadamente Seguro*. Cada solución consta de dos valores, uno por cada función de evaluación (F_r y F_s). Para cada uno de los algoritmos se muestra la mejor solución, así como la media de las soluciones y la desviación típica de las mismas (en las 30 ejecuciones). Los mejores resultados están marcados en negrita para poder identificarlos fácilmente. Para extraer conclusiones y analizar correctamente la tabla, es aconsejable centrar el estudio sobre los valores medios más que sobre los mejores, prestando igualmente atención a la desviación típica, pues ambos son los factores que

		Más Rápido (λ=0,9)		Más Seguro (λ=0,1)	
		F_r	F_s	F_r	F_s
CHAC-RTC	Mejor	**61,00**	244,90	74,00	27,30
	Media	66,42 ±3,29	225,19 ±90,26	84,68 ±4,89	28,36 ±0,48
CHAC-RTD	Mejor	66,50	295,20	82,50	28,00
	Media	73,30 ±2,98	249,61 ±57,79	94,22 ±5,90	29,27 ±0,59
CHAC-4-RTC	Mejor	66,00	285,20	81,00	28,00
	Media	71,70 ±3,70	316,66 ±58,73	98,13 ±15,99	108,46 ±63,79
CHAC-4-RTD	Mejor	64,00	304,90	81,00	28,00
	Media	67,18 ±1,76	296,56 ±25,39	90,13 ±4,77	28,85 ±0,44
MOACS	Mejor	64,00	304,90	77,00	27,60
	Media	70,77 ±2,43	294,66 ±79,44	93,60 ±6,93	29,23 ±0,68
BiAnt	Mejor	74,00	256,00	116,50	41,20
	Media	100,27 ±16,71	279,70 ±153,73	135,90 ±31,96	287,33 ±135,75
GRMO	Mejor	74,20	319,69	118,81	226,93
	Media	95,99 ±8,60	349,57 ±32,07	113,30 ±11,92	289,48 ±27,61
mono-CHAC	Mejor	72,00		**27,10**	
	Media	78,33 ±4,24		52,23 ±42,97	
		Extremadamente Rápido (λ=1)		Extremadamente Seguro (λ=0)	
		F_r	F_s	F_r	F_s
extr-CHAC-RTC	Mejor	**61,00**	244,90	73,00	27,20
	Media	**63,30 ±1,92**	273,08 ±57,67	82,07 ±5,40	**28,10 ±0,54**
extr-CHAC-RTD	Mejor	65,00	215,30	83,00	28,20
	Media	70,98 ±2,97	265,71 ±44,54	93,10 ±6,22	29,19 ±0,59
extr-CHAC-4-RTC	Mejor	62,50	584,50	88,00	28,70
	Media	68,70 ±4,03	451,29 ±172,51	94,52 ±13,59	118,07 ±70,14
extr-CHAC-4-RTD	Mejor	64,00	304,90	81,00	28,00
	Media	67,18 ±1,75	296,56 ±25,39	90,13 ±4,77	28,85 ±0,44

Tabla 7.5: Resultados para el Mapa PG Dos Enemigos y Ríos. (1500 iteraciones, 50 hormigas)

mejor describen el conjunto de las soluciones obtenidas.

A partir de los datos mostrados en dicha tabla, se pueden identificar varias comparativas y extraer diversas conclusiones.

Lo primero a destacar es un aspecto obvio de las soluciones y es que el valor de las mismas relativo al objetivo que goza de mayor prioridad, esto es F_r cuando se buscan caminos rápidos y F_s cuando se buscan caminos seguros, es siempre mucho mejor que el otro valor, dado que éste último se refiere al objetivo no prioritario, el cual se podría decir que no está siendo optimizado (como muestra se pueden observar las desorbitadas desviaciones típicas para dichos valores).

Otro aspecto a señalar a priori son las grandes diferencias que hay entre algunas soluciones referidas al coste en la función de seguridad (o coste en salud) F_s, tanto comparadas entre algoritmos, como para el mismo (a tenor de lo que indican la media y la desviación típica). La explicación se tiene

en el mapa (ver Figura 7.2), ya que la situación de los enemigos y el uso de armas hacen que si los caminos discurren por la zona central del mapa (a la vista e incluso sobre zonas letales), el coste en dicha función aumentará significativamente.

Estableciendo una comparativa entre algoritmos hay que señalar que, a priori, el algoritmo que mejores resultados ofrece para ambos objetivos es la implementación extrema de CHAC con la regla de transición RTC, es decir extr-CHAC-RTC, lo cual resulta bastante lógico dado que se trata de un algoritmo pensado para encontrar los mejores resultados para cada objetivo (ver Sección 7.3.1) ponderando para ello los términos que se refieren a él en las heurísticas y desechando los otros, y además asignándole una prioridad total por medio del parámetro λ, que como se puede ver es 1 o 0 respectivamente. Aunque existen dos detalles a resaltar a este respecto: por un lado, la implementación no extrema usando la misma regla, CHAC-RTC obtiene el mismo resultado como mejor solución, si bien mirando la media y desviación típica se podrá notar que éstas son peores que las del algoritmo extremo, por lo que dicha solución se deberá en gran medida al componente estocástico de que hacen gala estos algoritmos. Por otro lado, el algoritmo mono-CHAC sorprende en su desempeño por ofrecer la mejor solución en cuanto al coste en la función de seguridad (también referido como coste en salud), pero siguiendo el mismo modus operandi y razonamiento que en el caso anterior, podremos notar que su media es muy superior a la de otros algoritmos, por tanto nuevamente habrá sido fruto del azar la obtención de una solución tan buena.

Entrando en valoraciones generales, se podría hacer una comparativa entre los métodos que usan la RTC y los que usan RTD, siempre considerando los mismos algoritmos. A priori y viendo las definiciones de ambas reglas, se puede deducir que la RTC es una regla que aprovecha mucho más la explotación de soluciones conocidas, puesto que combina la información de todos los objetivos para crear uno solo, consiguiendo probabilidades absolutas que decantarán la búsqueda hacia los mejores caminos globalmente. Por su parte la RTD, al estar basada en la dominancia de vecinos y haber únicamente seis posibles, es más que probable que varios de esos vecinos compartan la misma probabilidad, pudiendo elegir uno u otro sin considerar realmente la bondad del camino hasta ellos. Esto incrementa el factor de exploración del método que haga uso de esta regla.

Mirando las soluciones obtenidas por unos (CHAC-RTC, CHAC-4-RTC, extr-CHAC-RTC y extr-CHAC-4-RTC) y otros métodos (CHAC-RTD, CHAC-

4-RTD, extr-CHAC-RTD y extr-CHAC-4-RTD), se da una situación curiosa: en el primero de los casos, el referido a los algoritmos CHAC, las implementaciones que hacen uso de la RTC siempre obtienen mejores resultados, pero en el caso de CHAC-4, son las implementaciones que utilizan RTD las que lo consiguen. La justificación de este efecto se puede encontrar en el número de objetivos con los que se trabaja en uno y otro caso, dado que cuantos más objetivos más posibles soluciones se podrán encontrar. Si consideramos que la RTC es una regla más explotativa, ésta será beneficiosa cuando el número de objetivos a combinar, así como el número de soluciones posibles no sea muy elevado. Por contra, si se tiene en cuenta que la RTD es una regla explorativa, su uso dará mejores resultados cuando los espacios de soluciones a estudiar sean mayores.

Centrando el estudio en las soluciones obtenidas por los métodos en base al número de objetivos con el que trabajan, se puede decir que la agrupación de los objetivos en uno solo (usando funciones agregativas), produce buenos resultados, pero no tan buenos como la división en dos o cuatro objetivos (obviando el resultado comentado anteriormente y centrándonos en los valores medios de las soluciones). Por su parte, la consideración de cuatro objetivos ofrece muy buenos resultados, sobretodo en el caso de los métodos que aplican la RTD, por las razones previamente expuestas y siendo bastante competentes en el caso de usar la RTC, si bien en estos casos sus resultados son peores que los obtenidos por el método CHAC usando la misma regla. Aunque también se debe notar que el uso de RTD en ese método resulta peor en general. La ventaja de los métodos con cuatro objetivos es que obtienen un FP mucho mayor (del orden de 10 veces mayor) y además bastante diverso, cuestión muy importante para un algoritmo multiobjetivo.

En cuanto a la comparativa entre los métodos propuestos y los métodos adaptados a partir de la bibliografía, éstos últimos han resultado ser bastante buenos considerando que han sido rediseñados para la resolución de este problema, siendo MOACS el mejor exponente de los mismos y obteniendo soluciones que mejoran en algunos casos a las de algunos de los métodos propuestos (CHAC-RTD y CHAC-4-RTC). BiAnt y GRMO han resultado ser un poco menos competitivos, si bien el primero tiene como justificación el no contar con el componente explotativo que añade el parámetro q_0 a los demás (por ser un SCHs) y haber sido comparado considerando el mismo número de ejecuciones. En el caso de GRMO ocurre algo parecido y es que al estar basado en dominancia padece las mismas taras que se comentaron anteriormente en cuanto a la regla RTD para dos objetivos, si bien además

hay que considerar que en su búsqueda de soluciones no cuenta con ninguna información de refuerzo, ni de un refinamiento iterativo de soluciones (las construye de una vez).

Entrando a valorar cada uno de los métodos:

- CHAC: obtiene los mejores resultados tras su implementación extrema y siempre considerando la RTC, cuando utiliza la RTD las soluciones empeoran debido al componente explorativo.
- CHAC-4: consigue resultados muy competitivos cuando aplica RTD y no tanto, aunque no son muy dispares si los comparamos con los demás métodos, cuando usa RTC.
- MOACS: tiene resultados más que aceptables, estando cerca de los mejores, si bien, en media se alejan de los algoritmos multiobjetivo propuestos.
- BiAnt: sus soluciones tienen costes demasiado altos, si bien es achacable con seguridad al alto factor explorativo que lo caracteriza y que se puede comprobar en la extrema desviación típica de sus resultados. Un aumento de su factor explotativo mejoraría dichas soluciones.
- GRMO: obtiene los peores resultados, si bien en media incluso mejora a BiAnt en algunas ocasiones. Se trata de una heurística muy básica.
- mono-CHAC: obtiene soluciones muy competentes, mejores que las de BiAnt y GRMO, aunque sensiblemente peores que las de los demás métodos propuestos.

A continuación se tienen varias figuras en las que se muestran de forma gráfica las mejores soluciones obtenidas por cada método, proyectadas sobre el escenario que se está analizando (MPG-2EneRios). Dicha proyección se ha hecho considerando la capa de información subyacente, ya que sobre el mapa real no se podría realizar un análisis con tanto detalle como sobre ésta. De esta forma se podrá visualizar la calidad de las mismas desde el punto de vista de un usuario que interactúa con el simulador mSS. Además se podrán analizar dichas soluciones, complementando o aclarando en algunos aspectos los datos que se muestran en la Tabla 7.5.

Las Figuras en cuestión son 7.5, 7.6, 7.7, 7.8, 7.9 y 7.10 y en ellas se muestran las soluciones agrupadas según métodos, dada la imposibilidad de mostrarlas todas juntas por cuestiones de visibilidad.

Como nota aclaratoria (ver Apéndice B4.2) señalar que los colores del borde de las celdas de los caminos solución indican la ocultación de dichas celdas respecto de los enemigos, teniendo éstas el borde negro cuando son vistas por algún enemigo y rosa cuando son ocultas a todos ellos.

La Figura 7.5 muestra las soluciones obtenidas para el algoritmo CHAC en sus dos implementaciones según considere las reglas RTC o RTD. Como se puede ver en ella, las soluciones correspondientes a los caminos más rápidos (a la izquierda) resultan ser rutas muy directas al destino, evitando en lo posible las celdas batidas por armas, así como aquellas que impliquen un mayor consumo de recursos, como agua o bosques (si bien muchas veces es inevitable atravesarlas). Como contrapartida, estos caminos discurren por zonas casi completamente vistas por los enemigos, lo que supondrá un alto coste en salud (en la función de seguridad), como se puede ver en la Tabla 7.5 en el caso de la solución más rápida, ya que ésta depende muy mucho de la visibilidad, la cual es muy penalizada.

En el caso de los caminos más seguros (a la derecha), se puede ver que éstos discurren por zonas completamente ocultas a los enemigos, dando un gran rodeo y obviando casi completamente en algunos tramos el dirigirse hacia el destino. Como se puede comprobar (ver ocultación de las celdas en la Figura 7.2), dicho camino es oculto en su primer tramo debido a que las celdas se encuentran tras (o dentro de) zonas boscosas. En su tramo final las celdas se encuentran fuera de la zona que determina la capacidad de adquisición de los enemigos, por lo que también son ocultas. Al igual que en el caso anterior, la menor consideración del otro objetivo (en este caso la rapidez), dando un gran rodeo, se traduce en un mayor coste en recursos (en la función de rapidez) que se puede ver nuevamente en la tabla 7.5 en el caso de la solución más segura.

Ambos comportamientos se han calificado como notables por los miembros del MADOC siguiendo las premisas de la táctica militar.

Mirando las soluciones relativas a la RTD en dicha figura, sobretodo la más segura, es posible notar que hay pequeños tramos que podrían haberse optimizado para ahorrar algunas celdas. Esto demuestra gráficamente que las soluciones de este método no son tan buenas como las del otro y que posiblemente sean debidas a un mayor componente explorativo.

Respecto a las soluciones mostradas en la Figura 7.6 para CHAC-4, las conclusiones son similares a las anteriores, si bien se puede notar que los caminos obtenidos usando RTD son ligeramente mejores que los de RTC, como ya se apuntó con los datos de la Tabla 7.5.

En el caso de la Figura 7.7, las soluciones obtenidas para MOACS demuestran ser bastante competentes (muy similares a las anteriores). Por contra las mostradas para BiAnt ratifican el hecho de que este algoritmo necesita un mayor factor de explotación, dado que en el caso se la solución más segura son palpables los múltiples tramos en los que se podrían excluir muchas celdas innecesarias (extras). Además de esto, dicho camino discurre por multitud de celdas de tipo bosque o agua y supera varios desniveles, todo lo cual no hace sino aumentar el consumo de recursos desmesuradamente y también el de salud, pues las celdas tienen un coste inherente en salud (bajas de no combate).

La Figura 7.8 muestra igualmente los resultados para GRMO, los cuales al igual que en el caso de BiAnt, 'pecan' de pasar por muchas celdas vistas en el caso del camino mínimo y de atravesar muchas celdas innecesarias en el caso del camino seguro, aumentando todos los costes en ambas soluciones. Mono-CHAC obtiene siempre una solución 'consenso', es decir, intentará ser rápida y segura, y en este caso ha tendido con más empeño hacia la seguridad. De hecho se trata de la solución más segura de todas, a tenor de lo visto en la Tabla 7.5.

Para terminar, los resultados extremos mostrados en las Figuras 7.9 y 7.10 son bastante buenos en general, salvo en los casos de CHAC-RTD y CHAC-4-RTC, en los que, como ya se vio en la tabla 7.5, éstos empeoran. En las figuras se puede comprobar que efectivamente en dichas soluciones se tienen celdas superfluas o que discurren por zonas con coste en recursos o salud mayores.

Un último aspecto a comentar se refiere a los tiempos de ejecución de los algoritmos, pues todos tardan entre uno y tres minutos, aunque destacan por su velocidad mono-CHAC y GRMO, los cuales no llegan al minuto para el número de iteraciones que se han realizado en estos experimentos.

7.4.3. Mapa Panzer General - Montañas

En esta sección se resolverá el escenario comentado en la Sección 7.1.2, al que nos referiremos en adelante como *MPG-Montanias*. De modo que nuevamente se han aplicado cada uno de los algoritmos propuestos y adaptados, con la reglas RTC y RTD y con las variaciones extremas. Se han resumido sus resultados en la Tabla 7.6 que se muestra a continuación y cuya interpretación es igual a la tabla anterior de resultados.

		Más Rápido (λ=0,9)		Más Seguro (λ=0,1)	
		F_r	F_s	F_r	F_s
CHAC-RTC	Mejor	**74,36**	352,66	80,53	**336,18**
	Media	76,43 ±0,99	352,39 ±8,98	81,66 ±2,49	354,61 ±11,86
CHAC-RTD	Mejor	77,85	382,60	86,39	363,58
	Media	83,99 ±2,20	398,93 ±15,80	85,87 ±2,99	388,55 ±15,51
CHAC-4-RTC	Mejor	75,99	365,25	82,75	360,59
	Media	84,33 ±5,81	398,48 ±32,09	88,88 ±6,45	395,40 ±29,07
CHAC-4-RTD	Mejor	77,37	352,83	77,37	352,83
	Media	80,44 ±1,68	381,01 ±13,03	84,37 ±3,35	368,16 ±8,25
MOACS	Mejor	79,15	378,63	85,31	351,86
	Media	84,45 ±2,73	388,24 ±15,40	87,09 ±2,27	382,93 ±17,44
BiAnt	Mejor	89,70	460,33	96,47	415,56
	Media	116,65 ±20,98	528,56 ±96,95	138,06 ±26,57	620,65 ±117,80
GRMO	Mejor	79,45	394,79	81,56	366,89
	Media	83,74 ±2,05	391,38 ±12,16	85,07 ±3,90	386,05 ±9,22
mono-CHAC	Mejor	76,70		339,34	
	Media	78,27 ±1,28		358,93 ±10,79	
		Extremadamente Rápido (λ=1)		Extremadamente Seguro (λ=0)	
		F_r	F_s	F_r	F_s
extr-CHAC-RTC	Mejor	**74,36**	352,65	80,53	**336,18**
	Media	**76,30 ±0,95**	350,43 ±7,73	80,91 ±1,65	**351,02 ±13,24**
extr-CHAC-RTD	Mejor	79,16	398,99	85,07	357,11
	Media	84,91 ±2,74	392,10 ±13,79	86,00 ±3,29	387,78 ±14,30
extr-CHAC-4-RTC	Mejor	77,95	384,63	80,96	354,89
	Media	84,32 ±4,45	403,69 ±23,11	91,49 ±7,92	412,29 ±41,01
extr-CHAC-4-RTD	Mejor	77,89	363,21	89,45	354,78
	Media	80,44 ±1,68	381,01 ±13,03	84,37 ±3,35	368,16 ±8,25

Tabla 7.6: Resultados para el Mapa PG Montañas. (1500 iteraciones, 50 hormigas)

Las consideraciones a priori sobre los datos de la tabla son las mismas que para el experimento anterior (ver Sección 7.4.2).

Lo más característico de este escenario es que no existen enemigos reconocidos en él, como ya se comentó y eso se refleja en la tabla. En ella se puede ver que el coste en salud (función de seguridad F_s) es siempre muy alto, sobretodo si se compara con el coste en recursos o con las soluciones del experimento anterior. La razón es que en este escenario no habrá celdas ocultas, todas tienen un valor de visibilidad asociado que se calcula como la media de las visibilidades para esa celda, desde todas las que la rodean en un radio igual a la capacidad de adquisición de la unidad (como se comentó en la Sección 5.1.1.1 al hablar de la función de ocultación, O). Por tanto, siempre habrá un coste asociado por visibilidad a cada celda del camino, el cual será penalizado en la función de evaluación.

Analizando los datos de la tabla, y con la salvedad de los peculiares resul-

tados relativos a la seguridad, vemos que se repiten los patrones comentados en el caso anterior. Es decir, nuevamente el algoritmo que ofrece mejores resultados es extr-CHAC-RT, aunque seguido muy de cerca por su homónimo no extremo CHAC-RT.

En la comparativa entre RTC y RTD, se repiten los esquemas, lo que significa que los métodos CHAC-RTCs son mejores que los RTDs para dos objetivos, pero no así para cuatro, dónde los CHAC-4-RTDs mejoran a los mismos que aplican RTC. La explicación es la misma que en el experimento anterior, siendo esto debido al componente explotativo de RTC, bueno para dos objetivos y peor para cuatro, y al componente explorativo de RTD que se comporta de la forma opuesta.

Continúa siendo más provechosa la división en dos objetivos, seguida por la consideración de cuatro y por último la agrupación en un solo objetivo.

Respecto a la valoración de los algoritmos adaptados, nuevamente MOACS se destapa como un buen método, ofreciendo soluciones bastante competentes. BiAnt vuelve a adolecer de un exceso de exploración (o una falta de explotación) lo que lo convierte en el que peores resultados da. GRMO por su parte si que muestra un buen comportamiento en este escenario, debido a que la falta de enemigo hace que el camino más directo sea una muy buena opción.

Entrando a valorar cada uno de los algoritmos:

- CHAC: nuevamente obtiene los mejores resultados, sobretodo en extremo y con RTC. Con RTD las soluciones empeoran debido al componente explorativo.
- CHAC-4: obtiene resultados buenos con RTD y no tanto con RTC.
- MOACS: consigue resultados aceptables, no muy lejos de los mejores.
- BiAnt: soluciones con altos costes debido a su falta de explotación.
- GRMO: en este caso se comporta bastante bien, dado que las soluciones directas son buenas en este escenario.
- mono-CHAC: obtiene soluciones bastante buenas, llega a un buen consenso en este escenario.

A continuación se muestran nuevamente varias figuras en las que se han proyectado sobre el escenario MPG-Montanias (sobre la capa subyacente)

las mejores soluciones obtenidas por cada método. De forma que se podrán analizar dichas soluciones, complementando los datos que se muestran en la Tabla 7.6. En dichas figuras se han marcado los obstáculos naturales (fronteras entre celdas que la unidad no puede atravesar).

Dadas las condiciones del escenario previamente expuestas, las soluciones seguras intentarán transitar por celdas que tengan una ocultación media respecto de las de su entorno lo mayor posible, es decir, en este escenario serían las cercanas a los grandes desniveles, marcados aquí como obstáculos naturales.

Las soluciones se pueden ver en las Figuras 7.11, 7.12, 7.13, 7.14, 7.15 y 7.16.

En este caso no habrá celdas ocultas, por lo que todas las del camino solución tendrán el borde negro.

En este caso, todos los resultados son bastante similares, formando caminos muy directos al destino los cuales sortean los obstáculos naturales como es lógico ya que no se pueden atravesar, pero pasando entre ellos o cerca porque esas diferencias de alturas tan pronunciadas hacen que las celdas cercanas permanezcan ocultas a muchas de las que las rodean, teniendo un mejor valor de ocultación.

Simplemente se puede notar la aparición de celda superfluas en los caminos de los algoritmos señalados anteriormente como peores (CHAC-RTD, CHAC-4-RTC, MOACS), con especial mención hacia BiAnt, dado que en este escenario queda totalmente patente su falta de explotación.

También es de señalar el buen rendimiento del algoritmo GRMO, pues en este escenario los caminos directos son muy buenos resultados.

Para concluir este análisis, los tiempos de ejecución son del mismo orden que en el caso anterior.

7.5. Conclusiones relativas a los experimentos

En este capítulo se han realizado experimentos consistentes en la aplicación de los algoritmos propuestos y adaptados descritos en los Capítulos 5 y 6, sobre dos escenarios modelados previamente y definidos al principio del presente capítulo.

Posteriormente se han analizado los parámetros y pesos a utilizar por

los algoritmos, determinando su influencia en la búsqueda de soluciones. Del mismo modo, también se han presentado las características/restricciones a considerar en los escenarios.

Además, se han diseñado variaciones para los algoritmos CHAC y CHAC-4, las cuales únicamente se centran en uno de los objetivos principales (rapidez o seguridad), por lo que se han llamado *algoritmos extremos*. Éstos han sido creados para obtener soluciones que sirvan como base comparativa para los demás algoritmos, pero que no son aplicables en la práctica puesto que se requiere un mínimo de prioridad para cada uno de los objetivos (se deben considerar siempre los dos).

Respecto a los resultados obtenidos, se ha determinado que las mejores soluciones (al margen de las extremas) las obtiene el algoritmo CHAC aplicando la regla RTC. CHAC-4 usando RTD también ofrece muy buenas soluciones y se destapa como una muy buena alternativa obteniendo frentes de pareto mayores y más diversos. Los algoritmos adaptados de la bibliografía producen resultados dispares, siendo MOACS el mejor exponente, con soluciones cercanas a los CHACs, y con BiAnt presentando una clara falta de factor explotativo, la cual se podría remediar con un mayor número de iteraciones, por ejemplo, pero que no se ha querido hacer en este estudio para realizar una comparativa en las mismas condiciones. GRMO es un algoritmo muy limitado, obteniendo buenas soluciones cuando el escenario se resuelve de forma bastante directa. Por último mono-CHAC también obtiene muy buenas soluciones, las cuales ofrecen un gran consenso entre rapidez y seguridad.

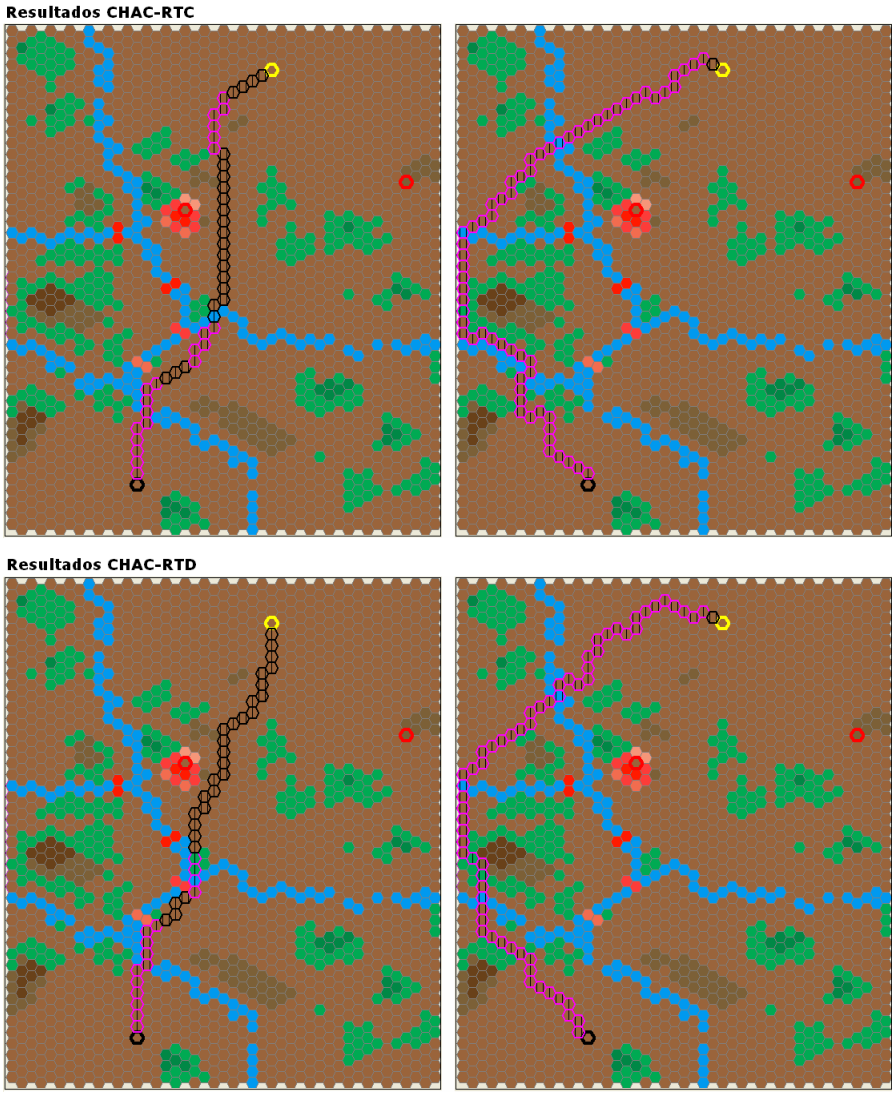

Figura 7.5: Resultados obtenidos por el algoritmo CHAC aplicando cada una de las reglas de transición (RTC y RTD), para el mapa MPG-2EneRios. A la izquierda se muestra la solución considerada como el camino más rápido y a la derecha el camino más seguro. Las celdas con borde rosa son ocultas a los enemigos y las de borde negro son vistas por alguno de ellos.

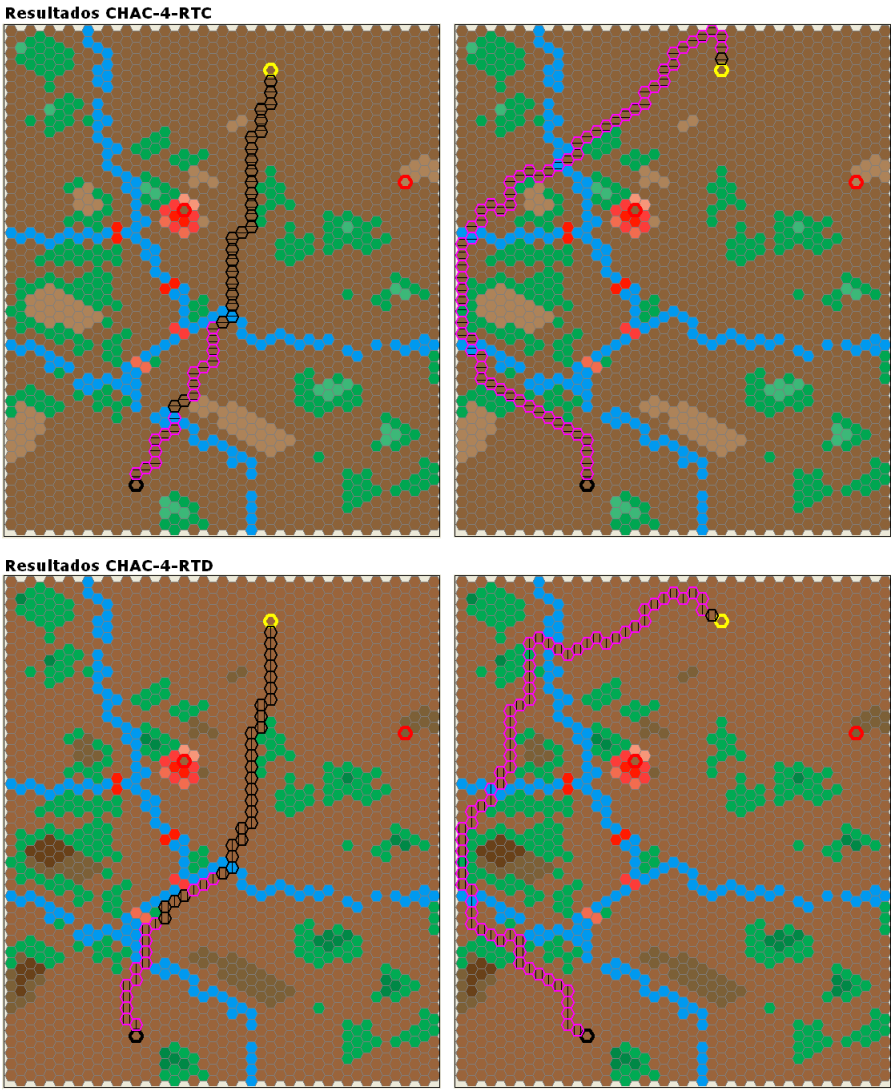

Figura 7.6: Resultados obtenidos por el algoritmo CHAC-4 aplicando cada una de las reglas de transición (RTC y RTD), para el mapa MPG-2EneRios. A la izquierda se muestra la solución considerada como el camino más rápido y a la derecha el camino más seguro. Las celdas con borde rosa son ocultas a los enemigos y las de borde negro son vistas por alguno de ellos.

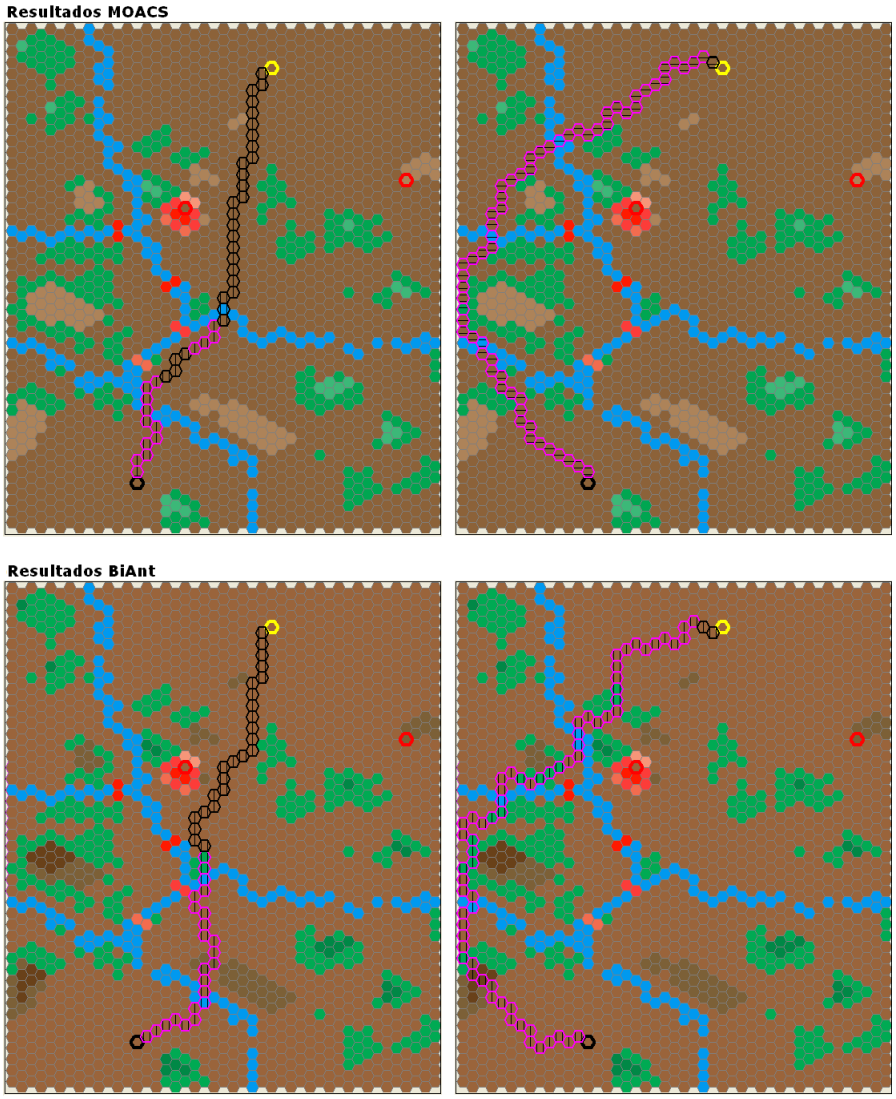

Figura 7.7: Resultados obtenidos por los algoritmos MOACS (arriba) y BiAnt (abajo) para el mapa MPG-2EneRios. A la izquierda se muestra la solución considerada como el camino más rápido y a la derecha el camino más seguro. Las celdas con borde rosa son ocultas a los enemigos y las de borde negro son vistas por alguno de ellos.

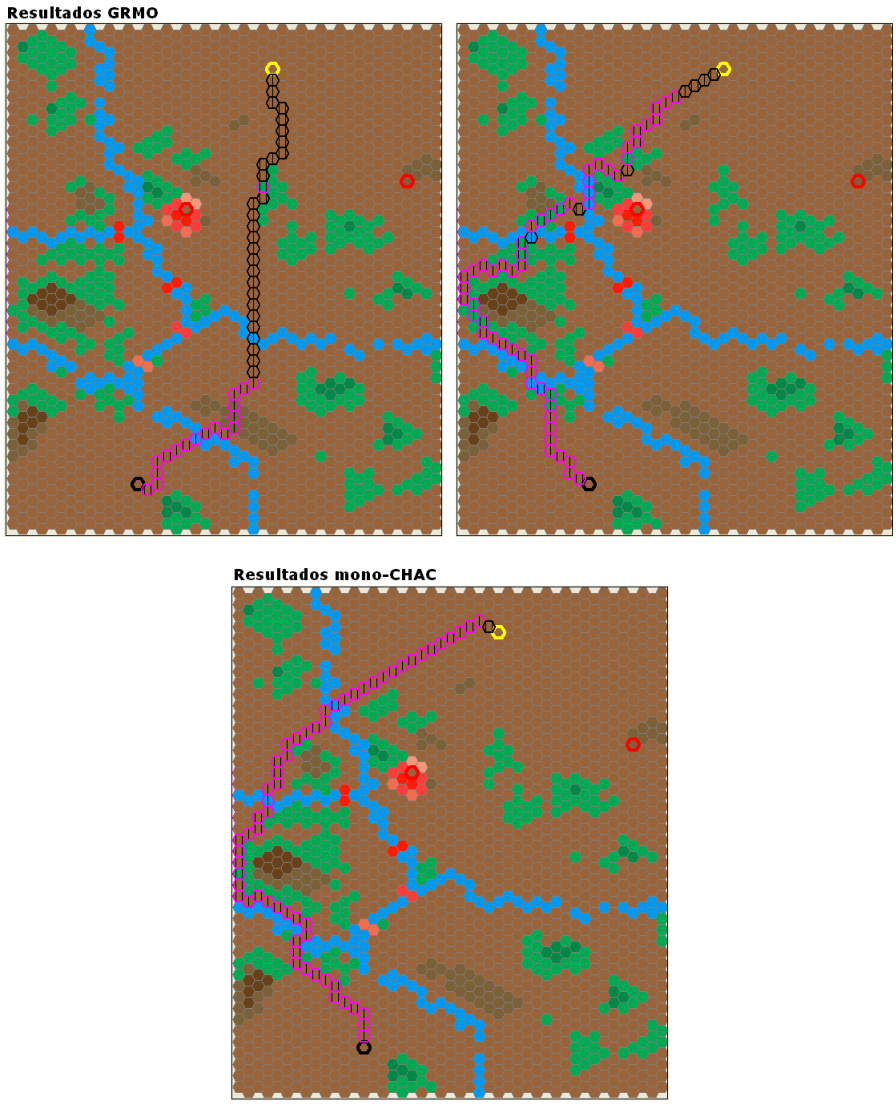

Figura 7.8: Resultados obtenidos por los algoritmos GRMO (arriba) y mono-CHAC (abajo) para el mapa MPG-2EneRios. A la izquierda se muestra la solución considerada como el camino más rápido y a la derecha el camino más seguro. Las celdas con borde rosa son ocultas a los enemigos y las de borde negro son vistas por alguno de ellos.

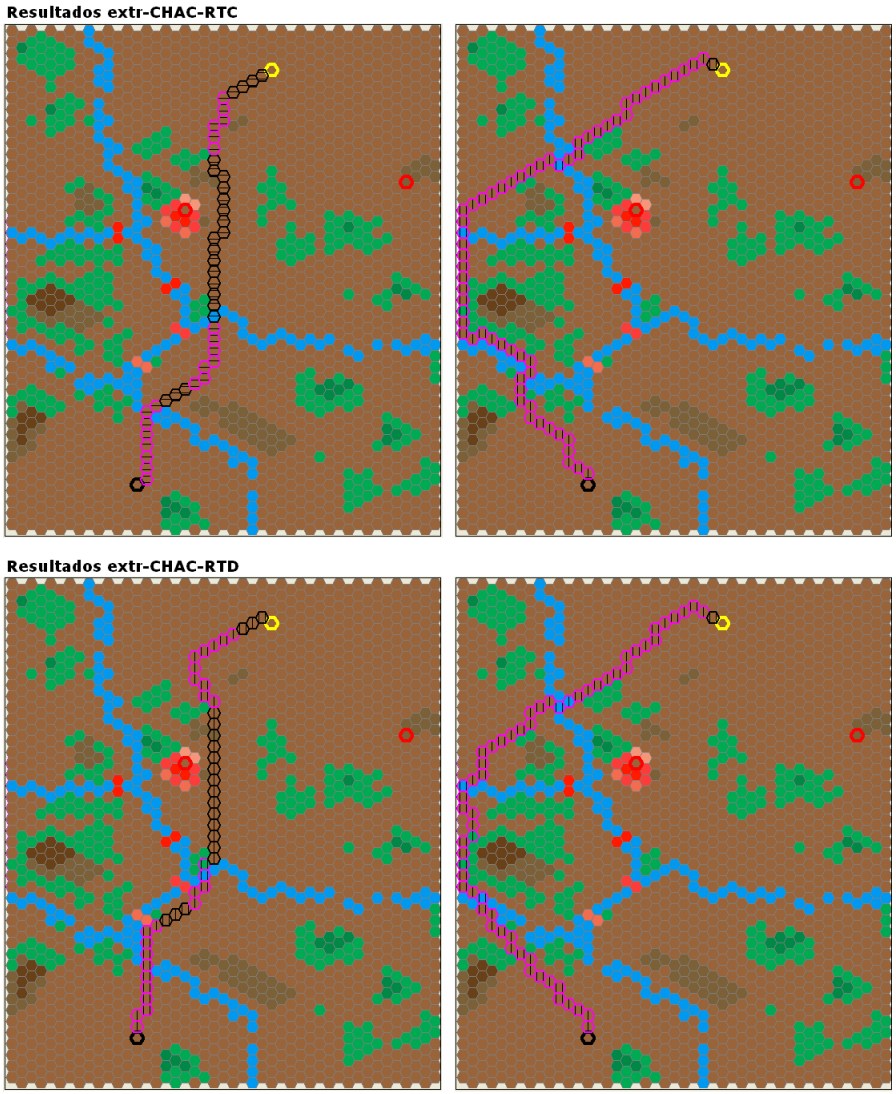

Figura 7.9: Resultados obtenidos por el algoritmo CHAC con configuración extrema (extr-CHAC), aplicando cada una de las reglas de transición (RTC y RTD), para el mapa MPG-2EneRios. A la izquierda se muestra la solución considerada como el camino más rápido y a la derecha el camino más seguro. Las celdas con borde rosa son ocultas a los enemigos y las de borde negro son vistas por alguno de ellos.

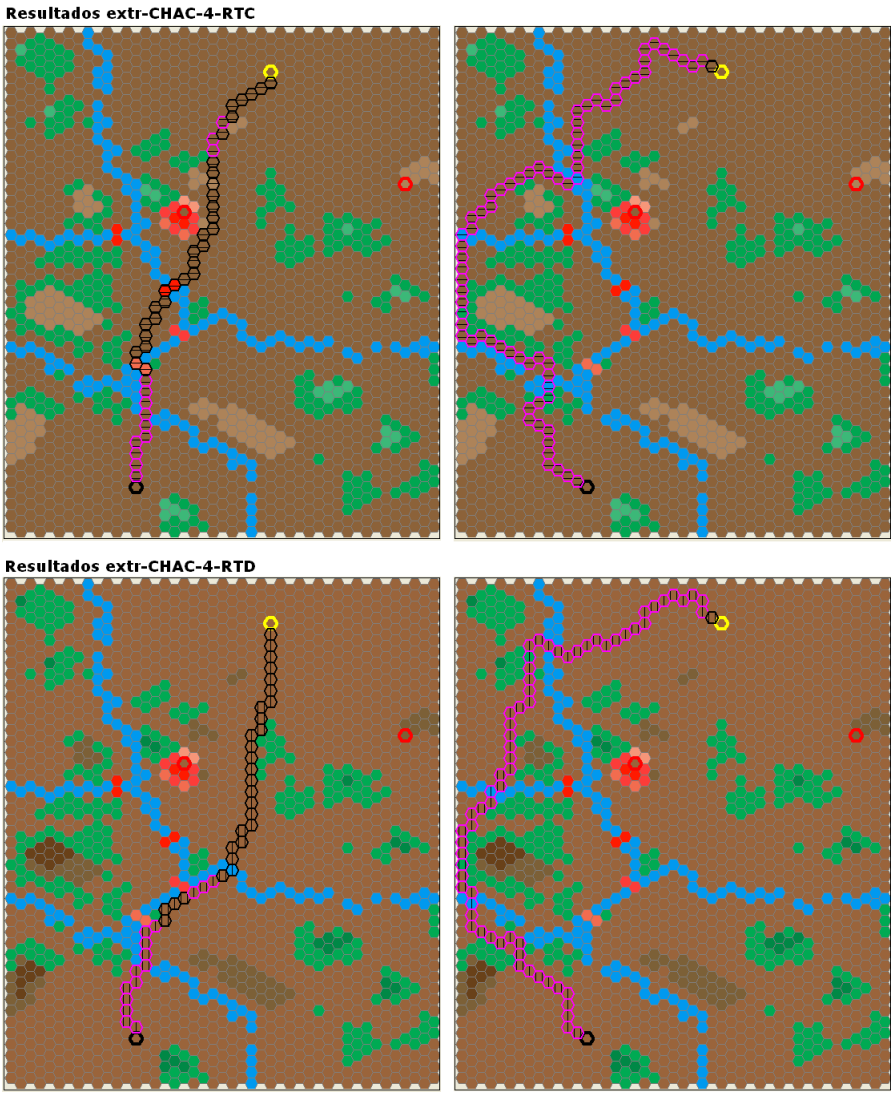

Figura 7.10: Resultados obtenidos por el algoritmo CHAC-4 con configuración extrema (extr-CHAC-4), aplicando cada una de las reglas de transición (RTC y RTD), para el mapa MPG-2EneRios. A la izquierda se muestra la solución considerada como el camino más rápido y a la derecha el camino más seguro. Las celdas con borde rosa son ocultas a los enemigos y las de borde negro son vistas por alguno de ellos.

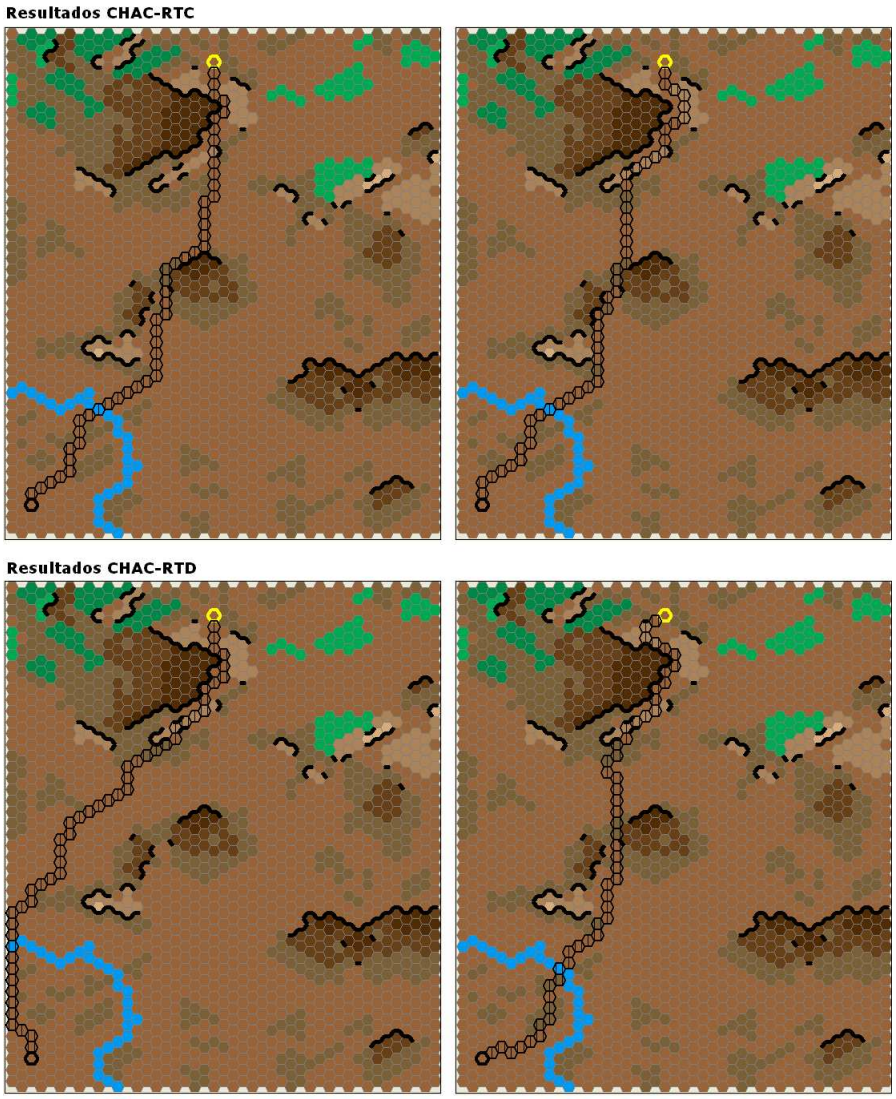

Figura 7.11: Resultados obtenidos por el algoritmo CHAC aplicando cada una de las reglas de transición (RTC y RTD), para el mapa MPG-Montañas. A la izquierda se muestra la solución considerada como el camino más rápido y a la derecha el camino más seguro.

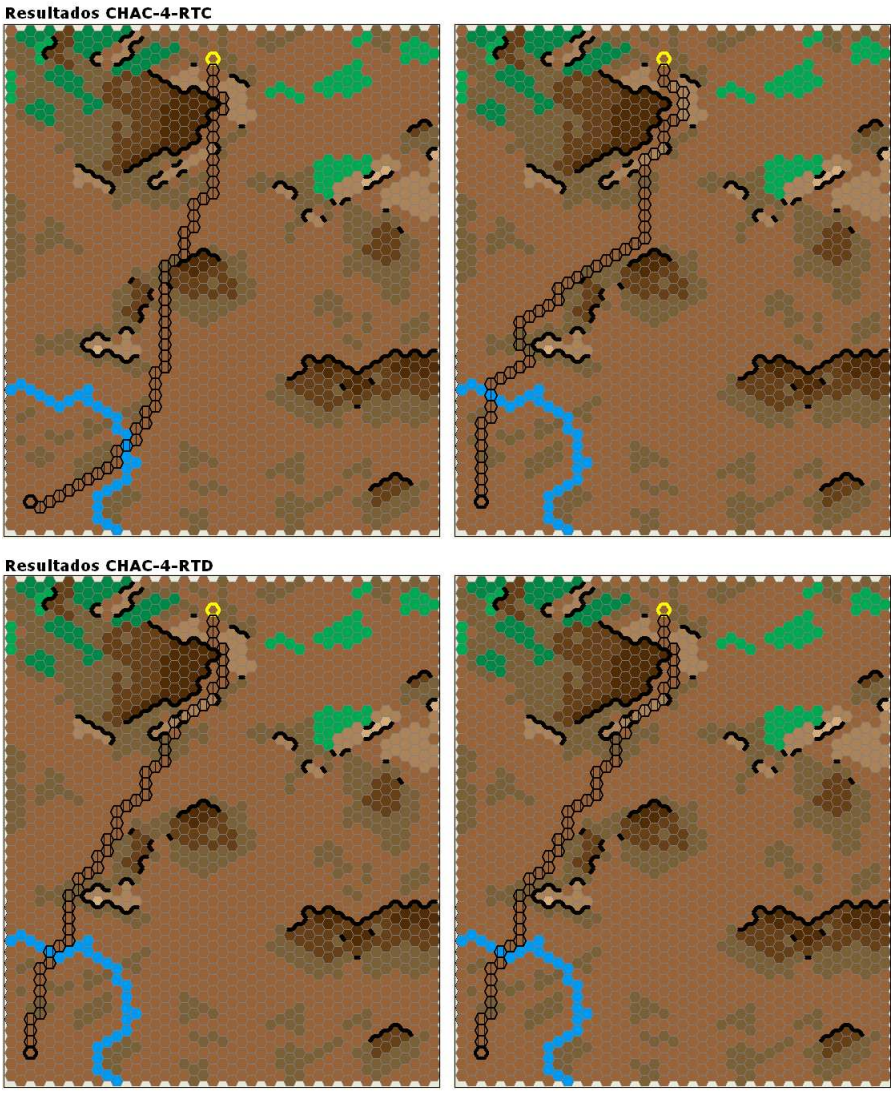

Figura 7.12: Resultados obtenidos por el algoritmo CHAC-4 aplicando cada una de las reglas de transición (RTC y RTD), para el mapa MPG-Montañas. A la izquierda se muestra la solución considerada como el camino más rápido y a la derecha el camino más seguro.

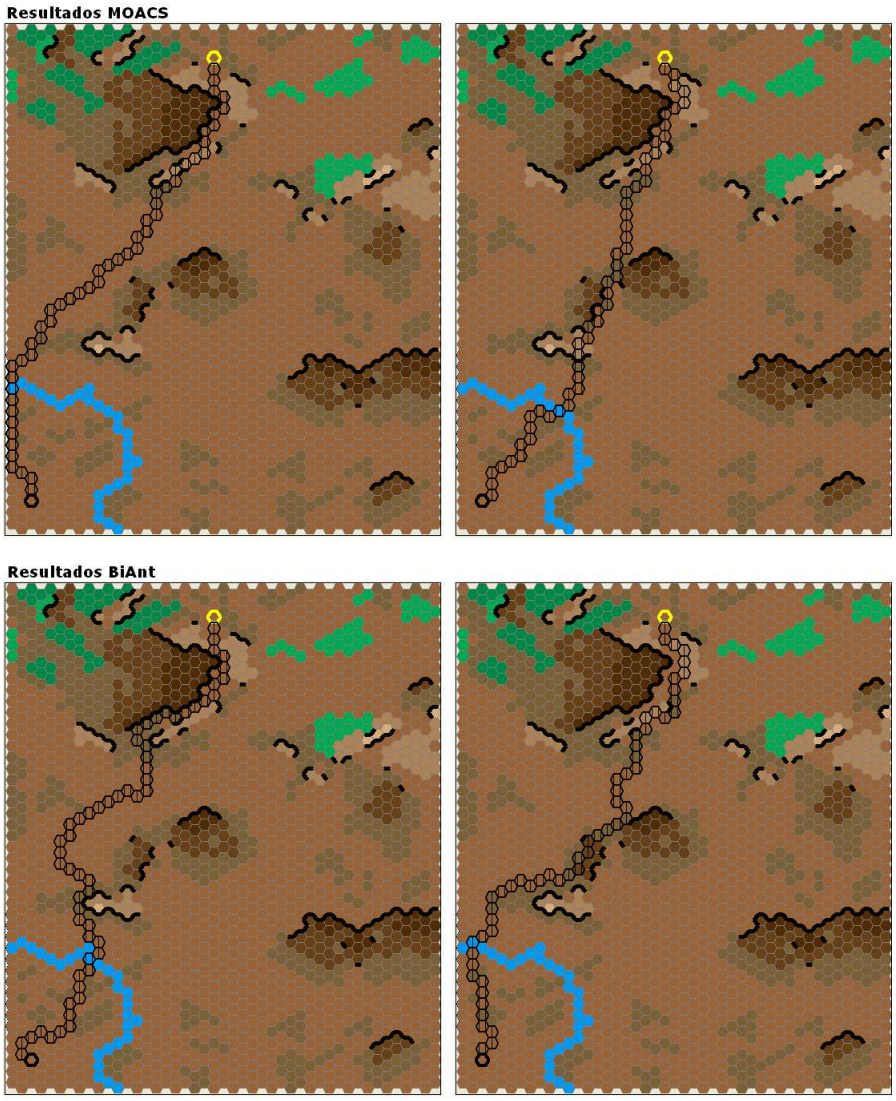

Figura 7.13: Resultados obtenidos por los algoritmos MOACS (arriba) y BiAnt (abajo) para el mapa MPG-Montañas. A la izquierda se muestra la solución considerada como el camino más rápido y a la derecha el camino más seguro.

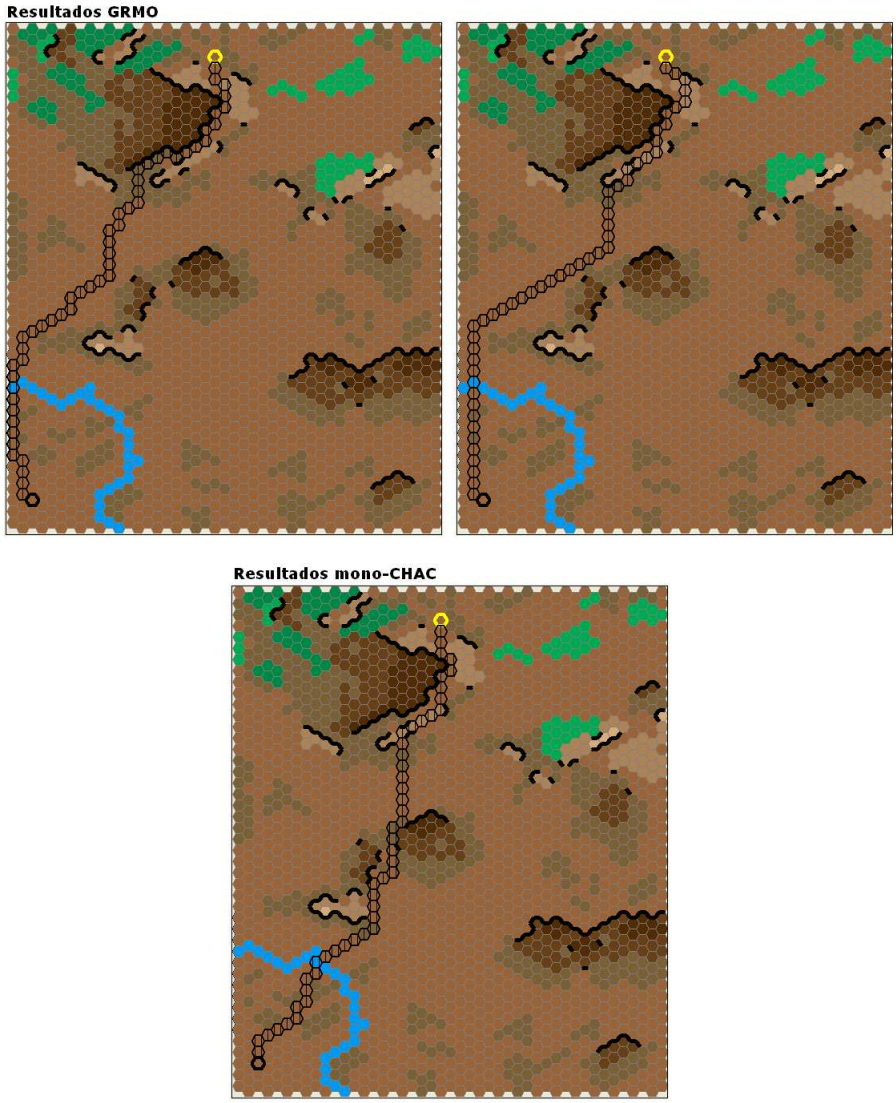

Figura 7.14: Resultados obtenidos por los algoritmos GRMO (arriba) y mono-CHAC (abajo) para el mapa MPG-Montañas. En el caso de GRMO, a la izquierda se muestra la solución considerada como el camino más rápido y a la derecha el camino más seguro.

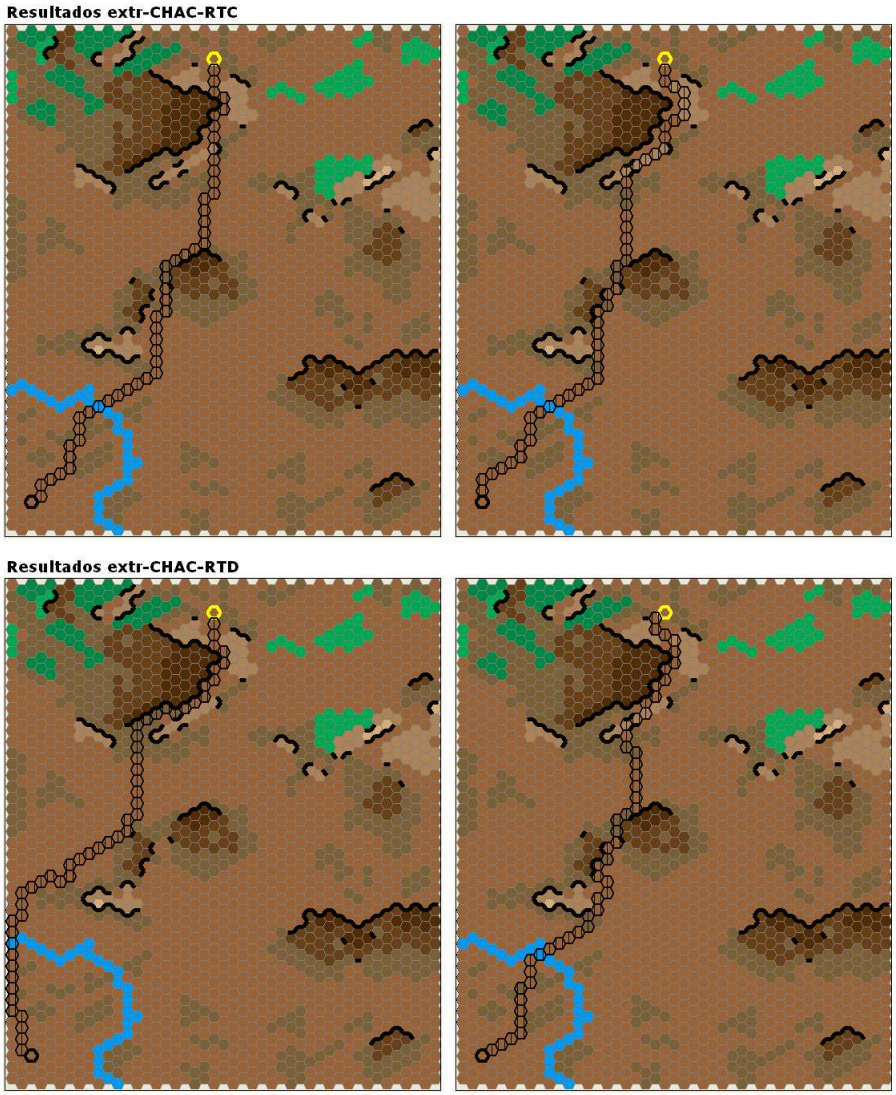

Figura 7.15: Resultados obtenidos por el algoritmo CHAC con configuración extrema (extr-CHAC), aplicando cada una de las reglas de transición (RTC y RTD), para el mapa MPG-Montañas. A la izquierda se muestra la solución considerada como el camino más rápido y a la derecha el camino más seguro.

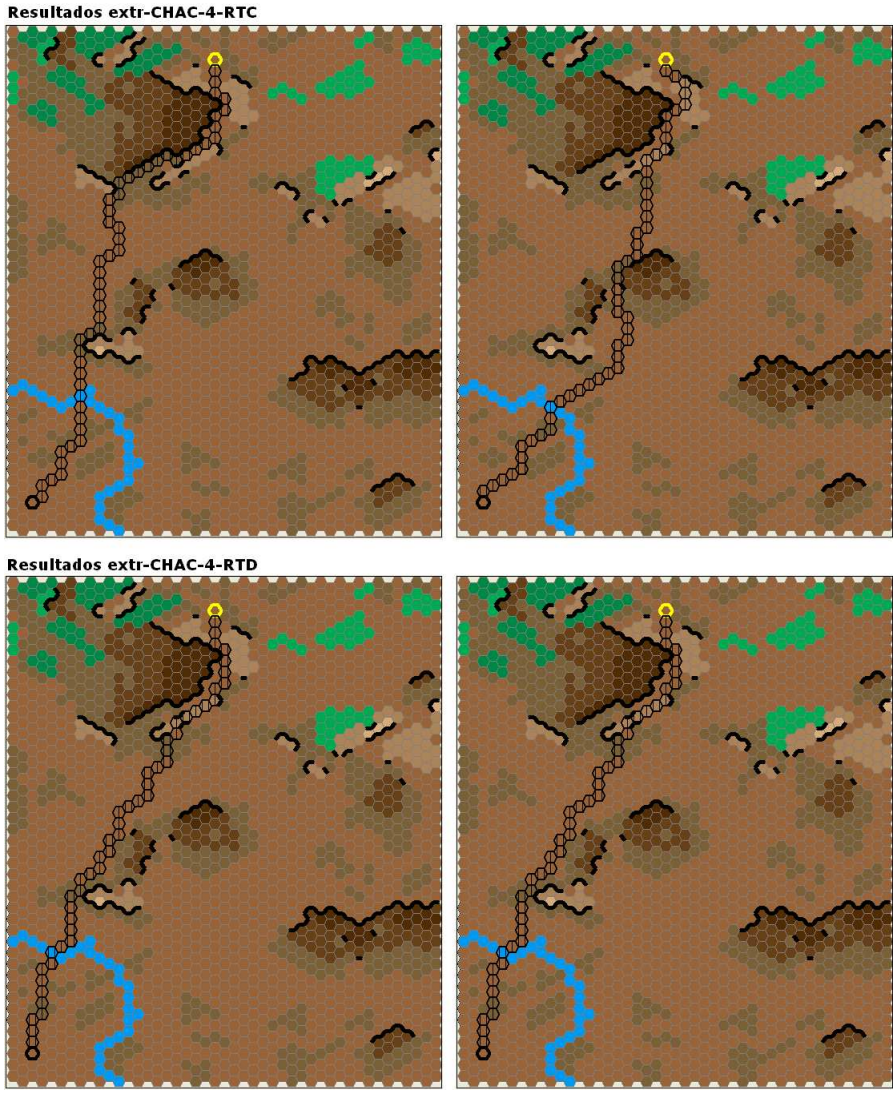

Figura 7.16: Resultados obtenidos por el algoritmo CHAC-4 con configuración extrema (extr-CHAC-4), aplicando cada una de las reglas de transición (RTC y RTD), para el mapa MPG-Montañas. A la izquierda se muestra la solución considerada como el camino más rápido y a la derecha el camino más seguro.

Capítulo 8

Conclusiones y Principales Aportaciones

En este trabajo se han estudiando y desarrollado varios métodos para la resolución de problemas de camino óptimo (PCOs) en los que se consideren varios criterios. Dichos problemas han sido localizados dentro de un entorno militar, por lo que han sido planteados como escenarios que modelan campos de batalla reales, en los que se debe mover una compañía militar entre un origen y un destino, considerando las propiedades del mapa así como los posibles enemigos presentes en el mismo y su influencia, tanto desde el punto de vista de la visibilidad, como por el uso de sus armas para batir zonas del escenario. Se ha considerado que los mapas tengan propiedades y restricciones a fin de hacerlos realistas. A su vez, en el problema se han definido dos objetivos a satisfacer, la *rapidez* y la *seguridad*.

Los métodos presentados en primera instancia han sido diseñados como algoritmos basados en colonias de hormigas para la resolución de problemas multiobjetivo (OCHMOs). Posteriormente se han estudiado y adaptado varios algoritmos existentes en la bibliografía a la resolución del problema comentado.

Se han definido varios escenarios realistas, planteando varios problemas, los cuales han sido resueltos usando estos métodos, estableciendo una comparativa entre los mismos.

Las principales aportaciones y conclusiones obtenidas quedan resumidas a continuación:

- Se han presentado tres métodos originales siguiendo el modelos de

los OCHMOs, bautizados bajo el nombre de **CHAC** (**C**ompañía de **H**ormigas **AC**orazadas), para relacionar el ámbito de algoritmos a que se refiere (algoritmos basados en colonias de hormigas) con el entorno en el que se resuelven los problemas (un marco militar). Éstos son:

  - *CHAC*: algoritmo que trabaja con problemas bi-criterio, considera una función heurística y una matriz de feromona por objetivo, y hace uso de dos posibles reglas de transición: la regla de transición combinada (RTC), basada en la combinación de la información de objetivos, y la regla de transición basada en dominancia (RTD), la cual aplica conceptos típicos de los problemas multiobjetivo a la elección del siguiente nodo en la construcción del camino solución, lo que resulta muy novedoso.
  - *mono-CHAC*: algoritmo que combina los dos objetivos principales en uno solo, mediante una función agregativa.
  - *CHAC-4*: algoritmo que subdivide los dos objetivos principales en otros dos cada uno, con lo que trabaja con cuatro objetivos de forma independiente. Para ello considera cuatro funciones heurísticas y cuatro matrices de feromona. Aplica igualmente la RTC y la RTD.

- Se ha definido un algoritmo genérico para la resolución de problemas con cualquier número de objetivos. Éste ha sido denominado *CHAC-N* y ha sido diseñado como generalización de los tres precedentes.

- Se han adaptado tres algoritmos existentes en la bibliografía a la resolución de PCOs con múltiples objetivos y concretamente al problema que se planteó en esta tesis. Estos métodos son:

  - *MOACS, Multi-Objetive Ant Colony System*: sistema de colonias de hormigas multiobjetivo propuesto inicialmente para la resolución del problema de enrutamiento de vehículos con ventana de tiempo.
  - *BiAnt, Bi-Criterion Ant*: sistema de hormigas bi-criterio para la resolución de problemas de programación de tareas en máquinas con pesos variables.

  - *GRMO, Greedy Multi-Objetivo*: algoritmo que sigue un enfoque voraz clásico, pero adaptado a la resolución de este problema. El cual se puede considerar como novedoso.

- Se ha realizado una modelización del PCO multiobjetivo, para situarlo en un entorno militar de forma realista, mediante la inclusión de propiedades y restricciones que lo acerquen a la realidad, así como funciones basadas en la simulación.

- A este respecto, se ha desarrollado un simulador, llamado mSS, como herramienta para la integración de dichas propiedades y funciones, y con el que es posible diseñar y resolver escenarios mediante los algoritmos planteados.

- Se realizado un análisis de parametrización de algoritmos, concluyendo cual es la influencia de cada uno de los parámetros en la búsqueda de soluciones y fijando los valores más convenientes mediante una experimentación sistemática.

- Se han estudiado varios escenarios realistas, modelados a partir de campos de batalla tomados de un videojuego de gran trascendencia y definidos mediante mSS.

- Se han diseñado varios métodos a partir de los algoritmos propuestos, que obtienen resultados extremos (considerando únicamente uno de los objetivos en la búsqueda), los cuales han servido como base comparativa.

- Se han resuelto los mencionados escenarios haciendo uso de todos los algoritmos y se han analizado y comparado sus resultados, concluyendo que CHAC haciendo uso de la regla RTC es el método que mejores soluciones obtiene (al margen de los métodos extremos). CHAC-4 utilizando la regla RTD también ofrece buenas soluciones y un frente de pareto más amplio y diverso.

- Los algoritmos adaptados obtienen soluciones competentes, excepto BiAnt, el cual necesita de un mayor factor de explotación para mejorar sus resultados.

- Las soluciones obtenidas por todos los algoritmos, han demostrado ser, según miembros militares a quien se ha consultado, bastante buenas

desde el punto de vista táctico. Por lo que se puede concluir que el problema ha sido modelado satisfactoriamente. Pudiendo considerarse dicha modelización como un nuevo aporte del presente trabajo.

- Todos los algoritmos han resultado ser bastante rápidos en su ejecución.

Finalmente, podemos afirmar que los objetivos propuestos al desarrollar esta tesis doctoral se han cumplido:

- Se ha diseñado una nuevo método genérico para la resolución de problemas de camino óptimo multiobjetivo.
- Se ha resuelto satisfactoriamente el problema militar de camino óptimo bi-criterio. Dado que las soluciones han sido contrastadas por expertos militares.
- Se ha creado un modelo fiel a la realidad y muy efectivo (según dichos expertos) de escenarios como campos de batalla.
- Se han adaptado exitosamente varios métodos ya existentes en la literatura, pero diseñados inicialmente para la resolución de otro tipo de problemas multiobjetivo. Uno de ellos, un enfoque greedy multiobjetivo ha sido diseñado de forma novedosa.
- Han sido analizados los parámetros utilizados por los algoritmos, así como las restricciones impuestas al modelo, y se han extraido conclusiones acerca de su influencia en aspectos de la búsqueda de soluciones.

Durante el desarrollo del presente trabajo, y directamente relacionados con él, se han remitido, a diferentes revistas y congresos nacionales e internacionales, las siguientes publicaciones:

- *(Revista Internacional)* A.M. Mora, J.J. Merelo, J.L.J. Laredo, C.Millán, J. Torrecillas. **CHAC. A MOACO Algorithm for Computation of Bi-Criteria Military Unit Path in the Battlefield: Presentation and First Results**. International Journal of Intelligent Systems. Aceptado para publicación en 2007.

- *(Congreso Nacional)* A.M. Mora, J.J. Merelo, C.Millán, J. Torrecillas. **Algunas Metaheurísticas Aplicadas a la Resolución de Problemas de Defensa**. I Simposio de Inteligencia Computacional (SICO 2005), dentro de CEDI 2005 Thompson, I. Rojas y H. Pomares Eds., pags. 169-176, Granada (España), Septiembre, 2005.

- *(Congreso Internacional)* A.M. Mora, J.J. Merelo, C.Millán, J. Torrecillas, J.L.J. Laredo. **CHAC. A MOACO Algorithm for Computation of Bi-Criteria Military Unit Path in the Battlefield**. I Workshop in Nature Inspired Cooperative Strategies for Optimization (NICSO'06), D. Pelta y N. Krasnogor Eds., pags. 85-98, Granada (España), Junio, 2006.

- *(Congreso Nacional)* A.M. Mora, J.J. Merelo, C.Millán, J. Torrecillas, J.L.J. Laredo, P.A. Castillo. **CHAC. OCHMO para la Resolución del Problema Bicriterio del Camino Óptimo de una Unidad Militar en el Campo de Batalla**. V Congreso Español sobre Metaheurísticas, Algoritmos Evolutivos y Bioinspirados (MAEB'07), F. Almeida, B. Melián, J.A. Moreno y J.M. Moreno Eds., pags. 207-214, Tenerife (España), Febrero, 2007.

- *(Congreso Internacional)* A.M. Mora, J.J. Merelo, C.Millán, J. Torrecillas, J.L.J. Laredo, P.A. Castillo. **Enhancing a MOACO for Solving the Bi-Criteria Pathfinding Problem for a Military Unit in a Realistic Battlefield**. First European Workshop on Evolutionary Computation in Transportation and Logistics (EVOTransLog), dentro de EVO* 2007, Lecture Notes in Computer Sciences, Mario Giacobini Ed., pags. 712-721, Valencia (España), Abril, 2007.

- *(Congreso Internacional)* A.M. Mora, J.J. Merelo, C.Millán, J. Torrecillas, J.L.J. Laredo, P.A. Castillo. **Balancing Safety and Speed in the Military Path Finding Problem: Analysis of Different Multi- and Mono-objective Heuristic Functions for an Ant-colony Optimization Method**. Workshop on Defense Applications of Computational Intelligence (DACI 2007), dentro de GECCO 2007, ACM Pres., Dirk Thierens et al. Eds., pags. 2859-2864, Londres (Inglaterra), Julio, 2007. **- Citado -**

- *(Congreso Internacional)* A.M. Mora, J.J. Merelo, C.Millán, J. Torrecillas, J.L.J. Laredo, P.A. Castillo. **Comparing ACO Algorithms**

for Solving the Bi-Criteria Military Pathfinding Problem. 9th European Conference on Artificial Life (ECAL 2007), Springer, Lecture Notes in Artificial Intelligence, F. Almeida et al. Eds., pags. 665-674, Lisboa (Portugal), Septiembre, 2007.

- *(Congreso Internacional)* A.M. Mora, J.J. Merelo, J.L.J. Laredo, P.A. Castillo, P.G. Sánchez, J.P. Sevilla, C. Millán, J. Torrecillas **hCHAC-4, an ACO Algorithm for Solving the Four-Criteria Military Pathfinding Problem**. Nature Inspired Cooperative Strategies for Optimization (NICSO 2007), Springer, Studies in Computational Intelligence Intelligence, G. Rudolph et al. Eds., vol. 129, pags. 73-84, Sicilia (Italia), Noviembre, 2007.

- *(Congreso Internacional)* A.M. Mora, J.J. Merelo, P.A. Castillo, J.L.J. Laredo, C. Cotta. **Influence of Parameters on the Performance of a MOACO Algorithm for Solving the Bi-Criteria Military Path-finding Problem**. IEEE Congress on Evolutionary Computation (CEC'08), dentro de WCCI 2008, IEEE Pres., pags. 3506-3512, Hong-Kong (China), Junio, 2008.

Habiendo sido citada una de dichas publicaciones, demostrando tener cierta relevancia dentro del área.

El desarrollo del presente trabajo ha permitido identificar una serie de temas y líneas de investigación originales, que se considera de interés abordar, a corto plazo, de la siguiente manera:

- Realizar un análisis más exhaustivo de la influencia de los parámetros en la búsqueda de soluciones, aplicando por ejemplo tests estadísticos más avanzados como ANOVA que permitan asignar los mejores valores a dichos parámetros.

- Hacer un estudio de la viabilidad de los resultados de MOACS y BiAnt, considerando un valor para el parámetro que pondera la prioridad de los objetivos λ que sea variable (uno para cada hormiga), como se usaba inicialmente en el esquema de dichos algoritmos.

- Hacer nuevos experimentos aplicando restricciones más acusadas, como niveles de recursos y salud muy limitados.

- Utilizar la configuración de parámetros que permita obtener las mejores soluciones en cada caso, añadiendo un mayor factor explotativo a los algoritmos que así lo requieran, a fin de determinar su verdadero potencial.
- Realizar experimentos en otros escenarios, con mayor dificultad para valorar las posibilidades de los algoritmos.

A medio y largo plazo se consideran de gran interés los siguientes temas:

- Aplicar CHAC a otros problemas bi-criterio, como por ejemplo el TSP bi-objetivo, para así comprobar su valía.
- Probar el algoritmo CHAC-N en algunos problemas multiobjetivo de distinto tipo, como pueden ser los problemas de enrutamiento de vehículos con ventana de tiempo.
- Incluir nuevas condiciones y restricciones al problema que se ha tratado, a fin de hacerlo más realista si cabe.
- Rediseñar el problema para incorporar dinamismo, es decir, enemigos que puedan cambiar de posición en el tiempo o propiedades variables del mapa que lo transformen cada cierto tiempo.
- Diseñar algoritmos que puedan trabajar con condiciones variables, como el dinamismo antes comentado.
- Diseñar algoritmos que puedan trabajar con restricciones de tiempo o con escenarios demasiado grandes para procesarlos como un todo, debiendo obtener soluciones parciales y partir de ellas para continuar su búsqueda.

APÉNDICES

A. El Simulador SIMBAD

En este apartado introduciremos brevemente algunas de las características del simulador real SIMBAD, utilizado actualmente en el Mando de Adiestramiento y Doctrina del Ejército de Tierra de España.

En él se pueden enfrentar mandos (jefes de unidades) del ejército entre si, haciendo uso de unidades reales (en cuanto a que tienen todas las propiedades y pueden realizar las mismas acciones que en la realidad), en campos de batalla reales (se usan datos cartográficos de un sistema de información geográfica). Estos enfrentamientos se hacen siguiendo fielmente la doctrina militar, tanto en lo referente a la estructura de las unidades, como a la táctica a considerar y a las reglas del combate a respetar. Durante la simulación, todo se rige por las mismas pautas que si se tratase de un combate real, por lo que esos mandos deberán seguir las órdenes de dirección de su superior. En concreto, SIMBAD está enfocado al entrenamiento táctico de mandos de nivel Batallón, es decir, con control sobre Compañías y dirigidos por un mando de Brigada. La apariencia del simulador se muestra en la Figura A1.

Como se ve en dicha figura, el simulador muestra la cartografía de una zona, pudiendo mostrarse los datos orográficos, curvas de nivel u otra de las distintas capas (del Sistema de Información Geográfica) que ofrezca el terreno en cuestión. En este caso aparecen señaladas las poblaciones de la zona, así como las carreteras (con sus diferentes tipos) y los ríos. También se pueden ver las unidades del bando azul, mostradas con sus símbolos tácticos, tratándose de tres compañías de carros de combate y el puesto de control de dichas compañías. En la imagen se muestran los datos del estado de una de ellas (se está moviendo a 20 Km/h y tiene sus efectivos al 100 %) a modo de ayuda contextual en amarillo. Además se pueden ver unas líneas en la parte baja de la imagen que son líneas tácticas puestas por el controlador de estas unidades para marcar algo.

En la Figura A2 se muestra un detalle de dichas unidades.

Cada unidad tiene asociadas una serie de acciones, según su tipo, sus efectivos y la situación en la que se encuentre. En la Figura A3 se pueden ver algunas de ellas.

El simulador permite a los usuarios ir manejando sus unidades siguiendo las normas de la doctrina militar, ofreciendo acciones, mermando efectivos, considerando propiedades, resolviendo combates, etc, de forma totalmente realista, pero sin ningún tipo de autonomía, requiriendo del controlador la

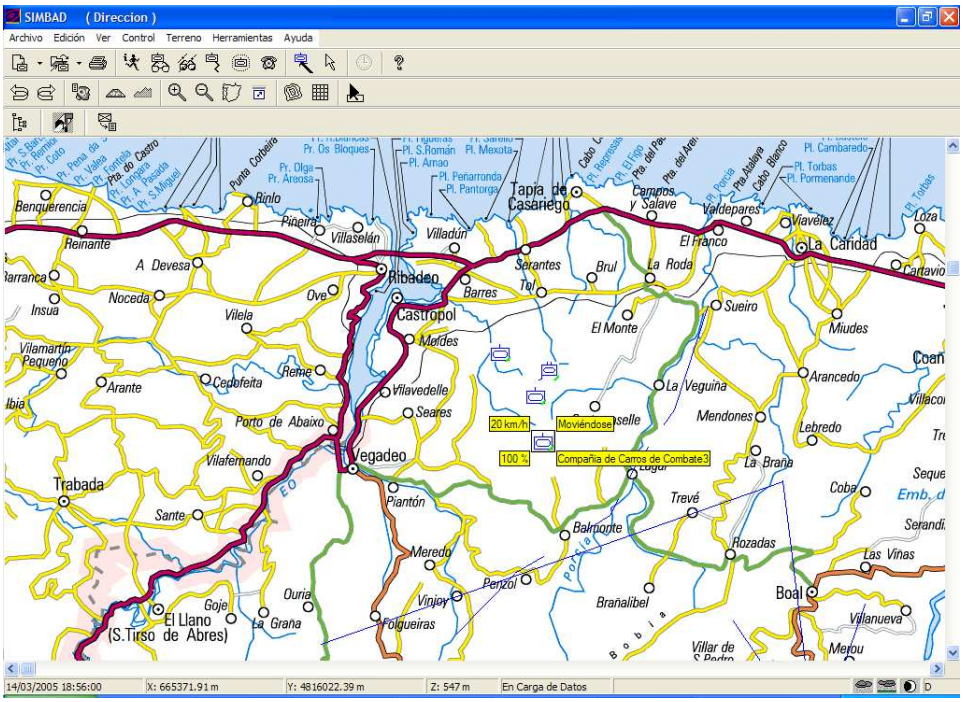

Figura A1: Simulador SIMBAD. Cartografía real (zona Asturias) con cuatro unidades del bando azul señaladas por sus símbolos tácticos.

toma de todas las decisiones.

B. Mini-Simulador SIMAUTAVA con grid hexagonal (mSS-HEXA)

B1. Introducción

En esta sección, se resumirán las principales características del simulador creado durante el desarrollo del presente trabajo.

En primer lugar habría que aclarar que este programa es considerado un simulador en cuanto a la funcionalidad que ofrece y a las reglas que define para la resolución de los problemas, pero no sirve para hacer simulaciones interactivas en tiempo real. De hecho fue creado como herramienta 'auxiliar' para el desarrollo y estudio de algoritmos.

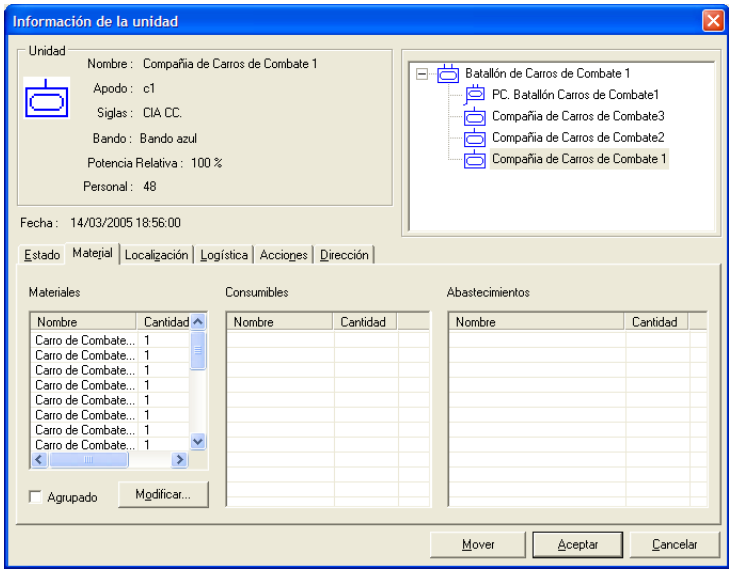

Figura A2: Datos asociados a las unidades en simulación, en este caso un batallón de carros de combate, formado por tres compañías y el puesto de control.

Figura A3: Algunas acciones asociadas a las unidades, según su tipo, situación y estado.

Por esa razón, sus posibilidades pasan por la *creación y edición de mapas*, que serían definiciones de escenarios que modelan un problema (búsqueda de camino óptimo en un campo de batalla, por parte de una unidad). Así como la *ejecución de varios algoritmos* para resolver dicho escenario, los cuales han sido desarrollados dentro del simulador. Por último permite la *visualización y estudio* de las soluciones obtenidas de forma intuitiva.

La apariencia de la pantalla una vez iniciado el programa se puede ver en la Figura B1.

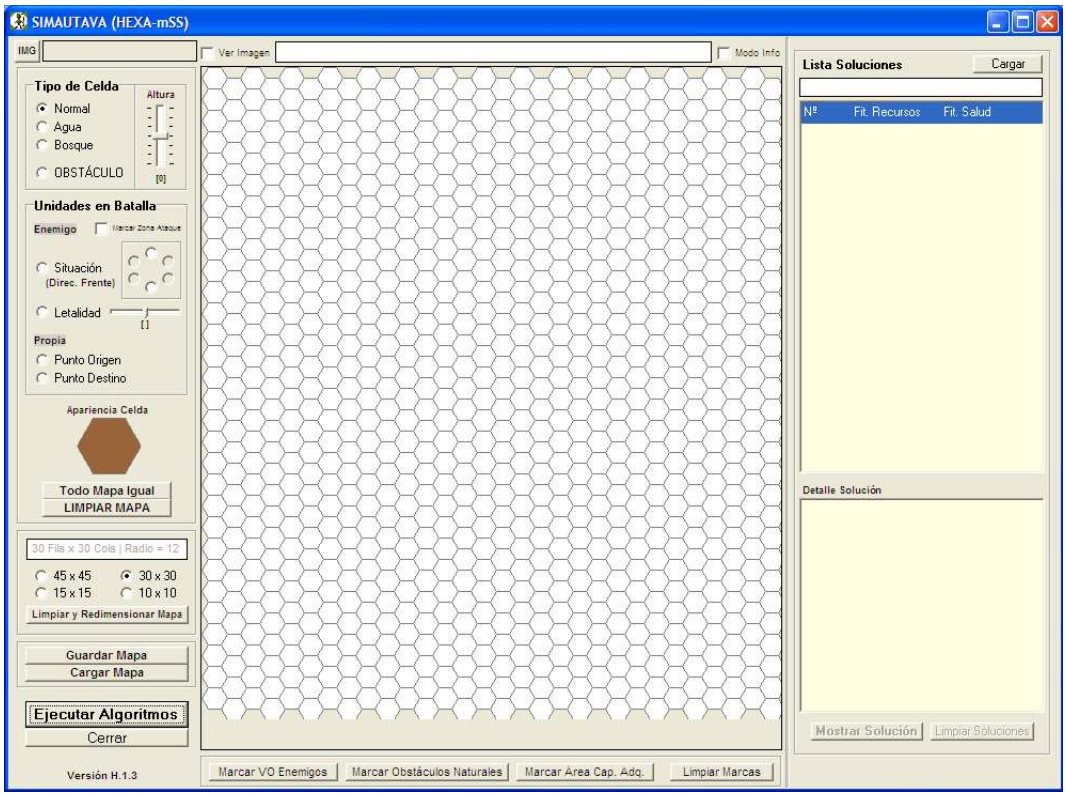

Figura B1: Pantalla inicial de mSS-HEXA (vers. H.1.3)

En ella que se puede ver en la parte izquierda las posibilidades de edición de mapas, en el centro el mapa con el que se está trabajando y en la parte derecha, las opciones en relación a la visualización y estudio de las soluciones obtenidas (por el algoritmo elegido sobre el mapa en cuestión).

Los escenarios, como se puede observar, consisten en una retícula de celdas

hexagonales de distintos tipos que definen un terreno (el campo de batalla). Sobre él se situarán los enemigos (si los hay), se marcarán las zonas abatidas por dichos enemigos (o simplemente dañinas para nuestra CIA) y, por último, se marcarán el punto origen del camino (situación actual de nuestra unidad) y el punto destino (punto que se desea alcanzar). Una vez definido un escenario, que será un problema del tipo que nos ocupa (búsqueda de camino óptimo), lo podremos resolver con los algoritmos implementados y visualizar las soluciones de forma gráfica e intuitiva.

Como apuntes decir que se va a considerar, como ya se ha comentado, una Compañía (CIA) como unidad del problema y como enemigos se considerarán Secciones (tres veces más pequeñas). El área de las celdas equivaldría a unos 500x500 m^2 en la realidad, es decir, se tendrían hexágonos con una apotema de unos 250 m y un lado de unos 300 m aproximadamente (ver Figura B2).

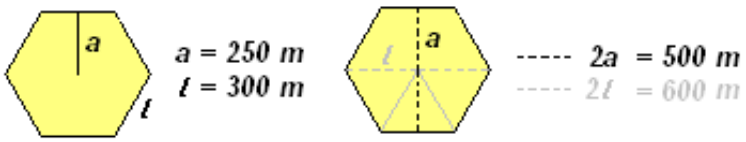

Figura B2: Dimensiones de una celda en base a las dimensiones reales en metros que se pretenden modelar. l es el lado y a es la apotema del hexágono.

La unidad dispondrá de una *salud (energía)*, correspondiente a los efectivos de que dispone y estado de los vehículos de la misma, y de unos *recursos*, correspondientes a combustible y otros consumibles. Ambas cantidades irán mermando a lo largo del camino en base a las penalizaciones que tengan asociadas las celdas según su tipo, diferencia de alturas o letalidad.

El programa ha sido desarrollado en Borland Delphi 7.

B1.1. Restricciones

Como restricciones se deben considerar:

- tanto la CIA, como cada unidad enemiga ocuparán exactamente una celda, ya que el tamaño de dicha celda equivale al tamaño del despliegue de una CIA en formación de ataque (unidad del problema) y de una Sección en formación defensiva (enemigos).

- una celda solo podrá ser de un tipo (y de un subtipo opcionalmente).

- podrá haber ninguno, uno o varios enemigos en el escenario.
- para definir completamente el problema es necesario indicar un punto origen y un punto destino, (solo uno de cada tipo).
- el camino solución no podrá atravesar celdas ocupadas por un enemigo ni obstáculos (naturales o artificiales). Tampoco podrá pasar más de una vez por una celda determinada.

B1.2. Terreno (Celdas)

El terreno ha sido modelado, como ya se ha visto, como una retícula de celdas hexagonales, dónde cada celda:

- Tiene unas *coordenadas* (X,Y), estando el origen en la esquina superior izquierda y creciendo la X hacia la derecha y la Y hacia abajo.
- Es de un *tipo*:
  - Normal (Tierra): terreno firme por el que circular sin problemas.
  - Agua: terreno pantanoso o con agua como ríos, lagos, etc.
  - Bosque: terreno con abundante y alta vegetación que dificulten el avance por él e incluso oculten a la unidad.
  - Obstáculo: terreno infranqueable para una CIA creado artificialmente (puede ser un agujero profundo, un muro, un campo de minas, etc).
- Tiene una *altura* entre -3 y 3, siendo 0 el nivel normal, positivo para elevaciones y negativo para depresiones.
- Tiene asociado un *daño por defecto* a la unidad (consumo de energía), justificado como bajas de no combate y averías de vehículos u otros aparatos.
- Puede tener una letalidad asociada (daño por impacto de armas enemigas o por otras causas).
- Tiene asociado un *consumo de recursos por defecto* a la unidad, que indica el combustible o cansancio que produce atravesar esa celda. Es un indicativo de la dificultad que conlleva cruzarla.

B1.3. Subtipos de Celdas

A la hora de definir completamente un escenario para el problema, es necesario indicar los puntos de origen y destino de la unidad, así como la situación de los enemigos (si los hay) y de las celdas afectadas por el impacto de sus armas. Para ello se consideran una serie de subtipos de celda:

- Enemigo: celda en la que está situada una unidad enemiga.
- Punto Origen: celda de la que parte la unidad (punto inicial del problema).
- Punto Destino: celda a la que quiere llegar la unidad (punto final del problema).
- Letalidad: se considera un subtipo y asocia a las celdas un daño (consumo de energía) por encontrarse batidas por la acción de algún arma del enemigo (o porque las queramos considerar dañinas para los efectivos de la unidad). El valor asociado a la celda estará entre 0 y 100, siendo 100 el valor más dañino.

NOTA: La letalidad por daño de las armas de un enemigo se obtiene como conjunción de tres factores, la probabilidad de que el enemigo abra fuego si la unidad se sitúa en la celda, la probabilidad de que el fuego alcance a la unidad y el daño que produciría de impactar sobre ella.

B2. Mapas en el Simulador

Como hemos comentado, uno de los objetivos del simulador es facilitar la definición de escenarios o mapas (problemas) para ser resueltos posteriormente mediante diversos algoritmos.

Los pasos para crear o editar un mapa son muy sencillos, pues basta con elegir el tamaño del mismo, posteriormente el tipo y subtipo de la celda a dibujar y hacer clic sobre la retícula del mapa en el punto que queramos que ocupe dicha celda. También es posible dibujar haciendo clic y manteniendo pulsado el botón, arrastrar el ratón para que se vayan dibujando celdas de dicho tipo (siempre que sea posible), al igual que se hace en cualquier programa de dibujo.

Si miramos nuevamente la Figura B1, podremos ver la apariencia inicial cuando no hay un mapa definido (ni cargado). Las celdas por defecto no

tendrán tipo, subtipo ni altura asociadas, por lo que será necesario definir todas las que hay en la retícula. A continuación se detallarán los posibles tipos y subtipos de celdas, así como todas las funciones asociadas a la edición que se pueden usar.

B2.1. Tipos y Subtipos de Celdas

Los tipos y subtipos que se consideran están comentados en los Apartados B1.2 y B1.3. Ambos son necesarios a la hora de definir completamente un escenario de problema o mapa. Existen varios radiobuttons en el simulador que nos permitirán seleccionar el tipo y subtipo de cada celda.

Los **tipos** son:

- Normal (Tierra): color marrón.
- Agua: color azul.
- Bosque: color verde.
- Obstáculo: color negro.

Del mismo modo, a cada celda se le puede asignar una **altura** dentro del rango [-3,3]. De manera que a mayor altura, más oscuro será el color del terreno y a mayor profundidad, más claro (manteniendo el mismo color del terreno), siguiendo el esquema de los extendidos tonos hipsométricos.

Los colores asociados a tipos y alturas se pueden ver en la Figura B3.

En lo que respecta a los **subtipos**, se representarán con bordes y formas de la celda:

- Enemigo: celda con color del tipo de terreno y altura correspondiente con borde muy grueso rojo.
- Punto Origen: celda con color del tipo de terreno y altura correspondiente con borde muy grueso negro.
- Punto Destino: celda con color del tipo de terreno y altura correspondiente con borde muy grueso amarillo.

La apariencia de los subtipos se puede ver en la Figura B4.

Además, será posible asignar un daño por impacto de armas a las celdas:

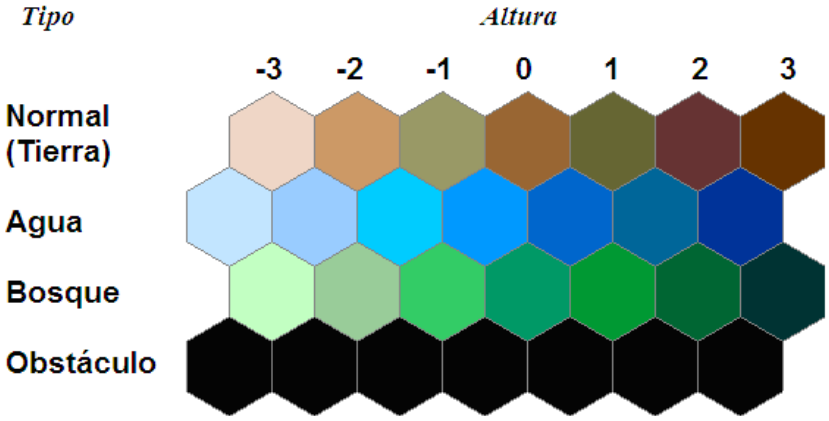

Figura B3: Colores asociados a los tipos y alturas en mSS-HEXA

Figura B4: Apariencia de las celdas en mSS-HEXA según sus subtipos

- Letalidad: las celdas se marcarán con distintos tonos de rojo en función del daño asociado (entre 0 y 100 como ya se indicó), según muestra la Figura B5.

Un ejemplo de mapa ya definido (en el que se han sobrescrito etiquetas para aclarar cada elemento) podría ser el que se muestra en la Figura B6.

En esta aplicación además, es posible crear un mapa a partir de imágenes reales, de manera que la definición de las celdas (tipos y subtipos), se podrán hacer para modelar dicho mapa real, editando directamente sobre la imagen del mismo y guardándose los datos 'debajo' (o alternando entre uno y otro con el checkbox *Ver Imagen*).

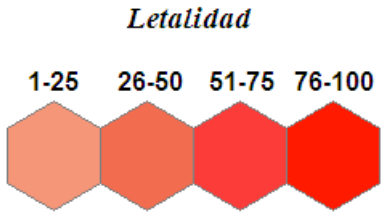

Figura B5: Niveles de letalidad y colores asociados a las celdas con impacto de armas en mSS-HEXA

De esta forma podremos modelar cualquier escenario real, como por ejemplo el que se muestra en la Figura B7.

B3. Herramientas de Visualización

En esta versión del mini-simulador, se han incluido tres herramientas que facilitarán la visualización de varias de las restricciones que se tienen en cuenta en la resolución del problema, en concreto, se ofrecen las posibilidades de marcar las celdas vistas y ocultas para los enemigos, marcar las celdas que determinan las capacidades de adquisición y marcar los obstáculos naturales para la unidad (ver Sección B5 para ver las definiciones de estos conceptos). Estas posibilidades se ofrecen como botones en la parte inferior de la pantalla principal (bajo el mapa).

B3.1. Marcar Celdas Vistas y Ocultas

Se usa el botón *Marcar VO Enemigos* y tras pulsarlo, se señalarán mediante un sombreado aquellas celdas que sean ocultas a todos los enemigos del mapa, bien por estar fuera de su capacidad de adquisición o por tener celdas que obstruyan la línea de visión (ver Sección B5) de los enemigos. En la Figura B8 se puede ver un ejemplo.

B3.2. Marcar Obstáculos Naturales

Se usa el botón *Marcar Obstáculos Naturales* y tras pulsarlo, se señalarán mediante una línea negra las fronteras entre celdas que la unidad no puede atravesar por haber una diferencia de altura entre ellas mayor de la que la

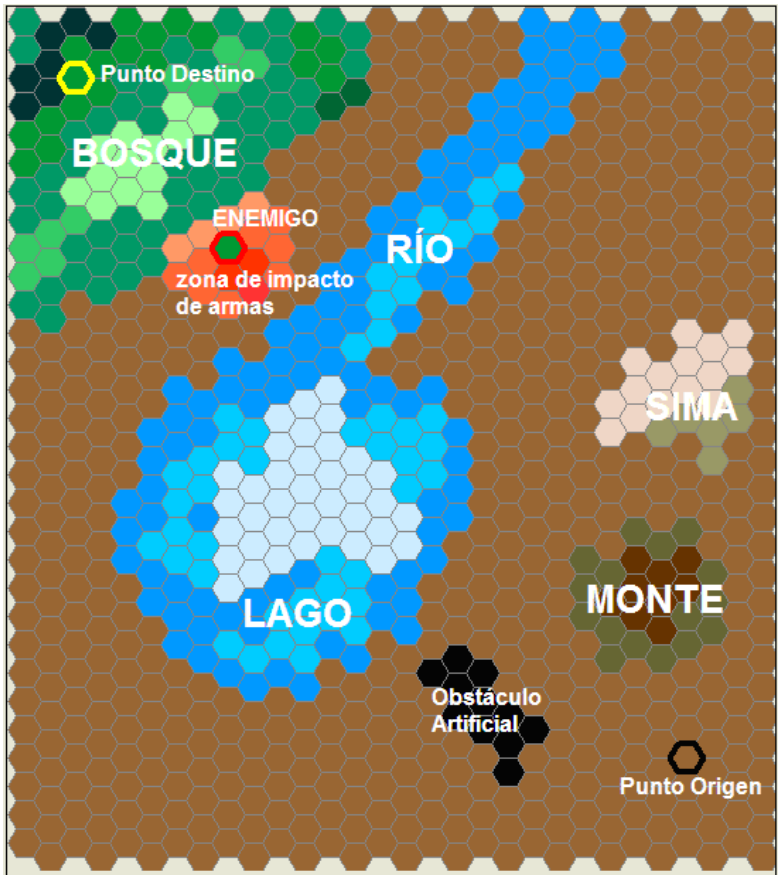

Figura B6: Mapa de ejemplos etiquetado para mayor claridad

unidad puede superar (por defecto 2). Un ejemplo de esta utilidad, se puede ver en la Figura B9.

B3.3. Marcar Área de Capacidad de Adquisición

Se usa el botón *Marcar Área Cap. Adq.* y tras pulsarlo, se señalarán con tramas de color las celdas que están incluidas en el área de capacidad de adquisición. La de los enemigos se marcará con trama roja y la de la unidad con trama amarilla. En la Figura B10 se puede ver un ejemplo en el que se considera 10 celdas como radio del área de enemigos y unidad del problema.

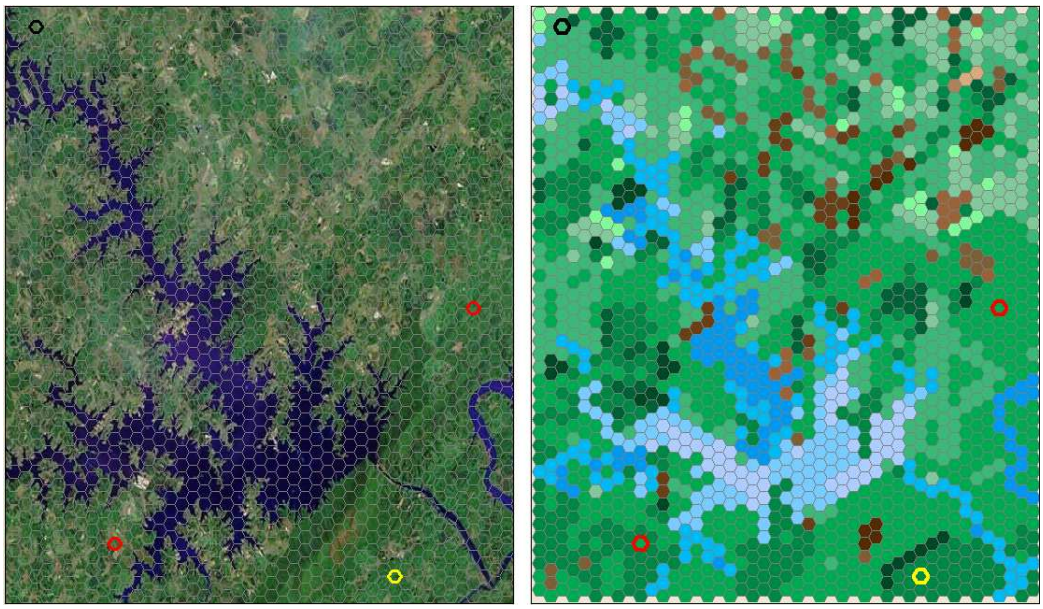

Figura B7: Ejemplo de modelado de un mapa real en el que se tienen un lago, ríos y mucha vegetación. A la derecha el mapa real y a la izquierda la capa de información inferior

B4. Visualización de Soluciones

Una vez haya terminado la ejecución del algoritmo elegido y si ha obtenido soluciones, se cargará en la parte derecha de la pantalla principal de mSS-Hexa la lista de soluciones encontradas.

B4.1. Datos de Soluciones

Una vez cargado el fichero de soluciones (o tras acabar la ejecución actual), se mostrarán los datos de las mismas, ordenadas según el criterio que predomine. Entre dichos datos se podrá ver el coste asociado a cada objetivo (salud/seguridad y recursos/rapidez) de la solución, para poder valorarlas, aunque según la teoría de los problemas multiobjetivo, ninguna será mejor que otra (el usuario elegirá la que mejor le parezca si es que tiene que elegir una), ya que las que mejoren en un objetivo empeorarán en el otro. La pantalla tendrá el aspecto mostrado en la Figura B11.

Como se puede ver en la Figura B11, en la parte superior de la lista de

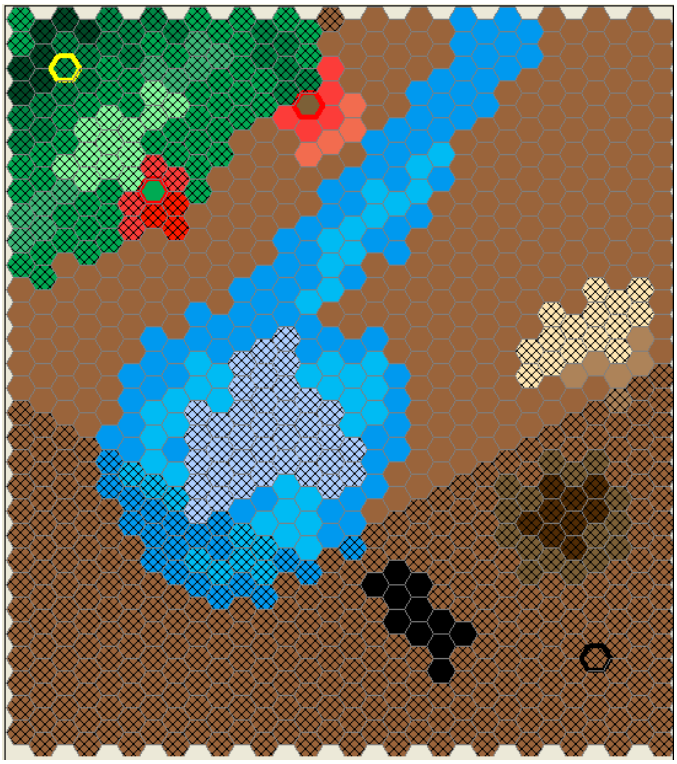

Figura B8: Ejemplo de marcado de celdas vistas y ocultas (con sombreado) a los enemigos

soluciones, se muestra el criterio con el que se buscaron dichas soluciones, es decir la relación de rapidez y seguridad con la que se buscó.

Los costes que se muestran en la lista superior no son estrictamente la salud y recursos consumidos, sino unos costes asociados a la solución que tienen en cuenta ambos factores, pero también la visibilidad de las celdas del camino desde las posiciones enemigas, sobretodo en el caso del camino más seguro (en el que penalizan mucho las celdas vistas por los enemigos).

La solución se mostrará como celdas con los colores del tipo y altura y con el borde negro en las vistas y rosa/fucsia en las ocultas. Esto se comentará en el Apartado B4.2.

NOTA: En un problema multiobjetivo suelen obtenerse miles de soluciones y este no era una excepción, pero por comodidad para el usuario se han omitido todas que tuvieran el mismo coste (serían prácticamente iguales), una celda en lugar de otra y todas las

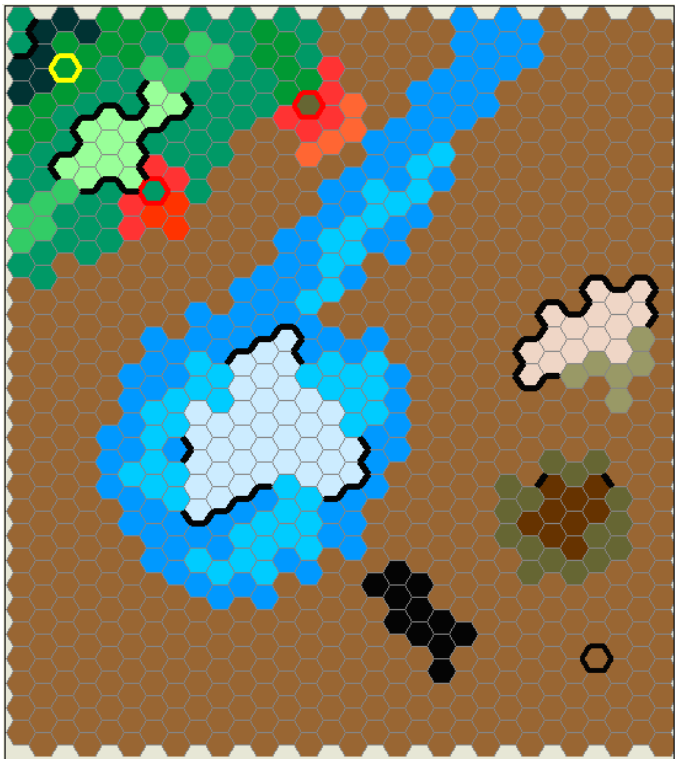

Figura B9: Ejemplo de marcado de obstáculos naturales (líneas negras gruesas) que la unidad no será capaz de atravesar

combinaciones posibles de este caso y solo se muestra una de ellas.

B4.1.1. Lista de Soluciones Como se puede ver en la Figura B11, las soluciones se muestran en la parte derecha de la pantalla principal a modo de lista. En dicha lista aparecen tres valores:

- *número de solución*: para poder identificarla.
- *Fitness Recursos*: coste de la solución en cuanto a rapidez. Se consideran tanto los recursos consumidos, como la visibilidad de la celda desde las posiciones enemigas (pero esto en poca medida) en las celdas del camino.
- *Fitness Salud*: coste de la solución en cuanto a seguridad. Se consideran

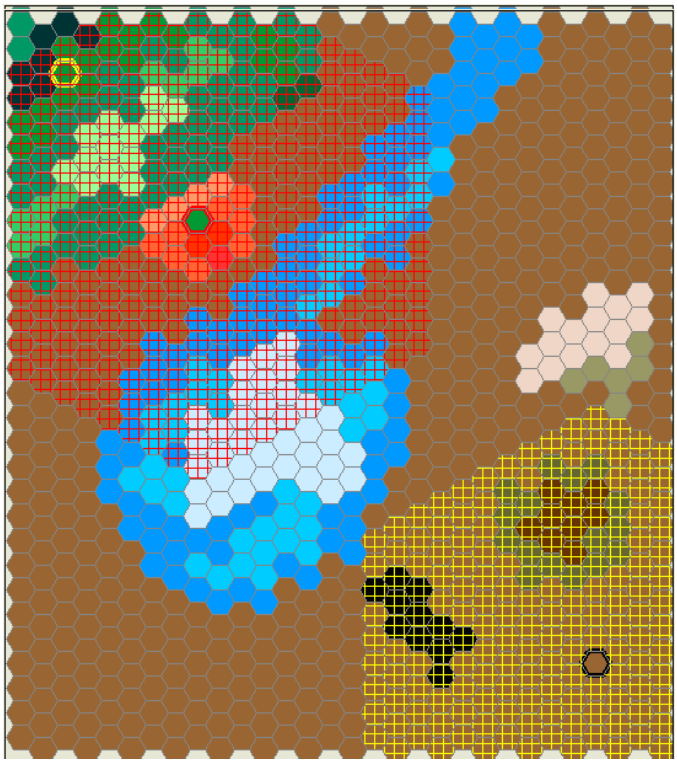

Figura B10: Ejemplo de marcado de capacidad de adquisición para la unidad (sombreado amarillo) y enemigos (sombreado rojo)

tanto la salud consumida (bajas de no combate y letalidad), como la visibilidad de la celda desde las posiciones enemigas (en gran medida) en las celdas del camino. La visibilidad es muy importante en el coste en seguridad porque las celdas vistas son susceptibles de ser atacadas por el enemigo y, por tanto, mucho más inseguras.

B4.1.2. Detalle de Soluciones Al seleccionar una solución de la lista (cuadro superior de la parte derecha), en el cuadro inferior se muestran sus detalles (ver Figura B11) estos detalles son:

- número de solución
- recursos y salud consumida en el camino (en esta ocasión si es la suma de recursos y de salud real que se consume)

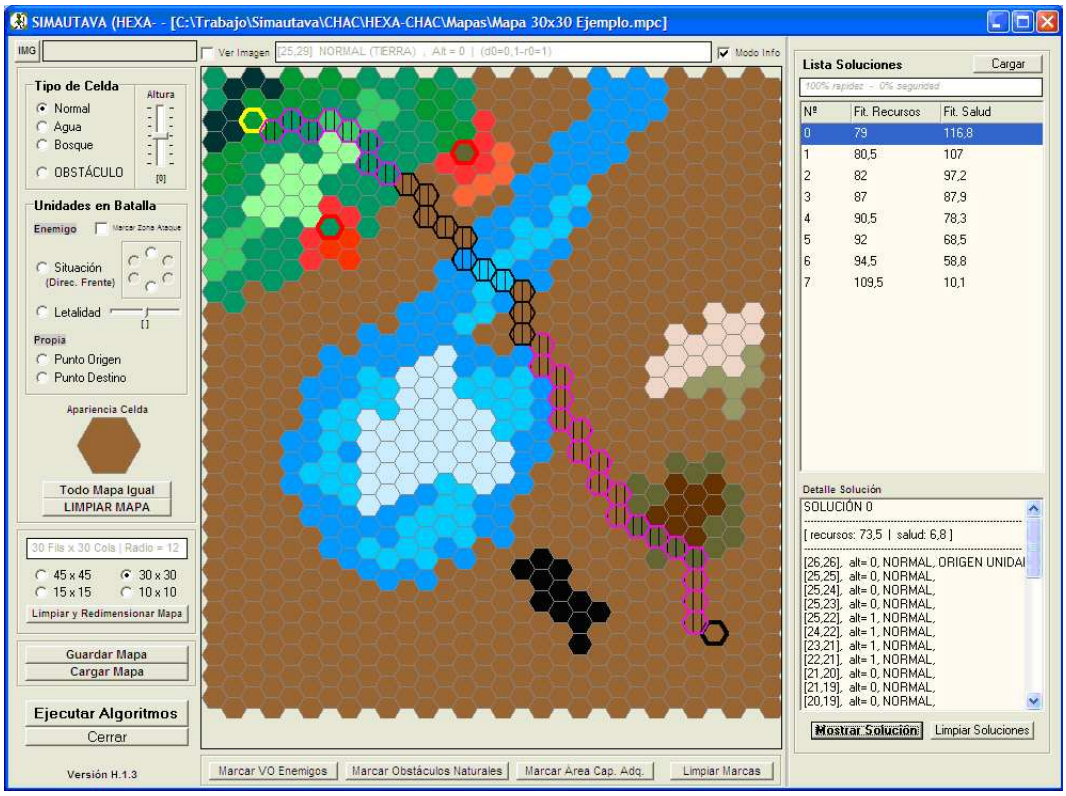

Figura B11: Ejemplo de una solución marcada sobre el mapa resuelto. Las celdas con borde rosa son ocultas a los enemigos y las de borde negro son vistas por alguno de ellos

- lista de celdas del camino desde el origen al destino. Se muestran los detalles de cada celda, coordenadas, tipo, altura, subtipo, etc.

B4.2. Descripción de Visualización de Soluciones

Una vez se tenga seleccionada una solución (y se muestren sus detalles en el cuadro inferior), es posible verla representada sobre el mapa, para ello bastará con pulsar el botón *Mostrar Solución* (debajo del cuadro de detalle de soluciones).

Tras eso, se dibujará dicha solución sobre el mapa que se tenga cargado (es importante asegurarse de que el fichero de soluciones cargado es el

correspondiente a ese mapa).

El camino estará formado por celdas *con borde grueso* que tendrán el color correspondiente al tipo y subtipo que tuviese esa celda en el mapa normal. El borde tendrá diferentes colores y grosores:

- *negro y muy grueso*: celda origen del camino
- *amarillo y muy grueso*: celda destino del camino
- ***negro grueso***: celda del camino que es vista desde alguno de (o todos) los enemigos
- ***rosa grueso***: celda del camino que es *oculta a todos* los enemigos (tienen obstaculizada su línea de visión hasta esa celda)

Es posible visualizar diferentes soluciones al mismo tiempo, a fin de poder comparar varias entre si del mismo algoritmo e incluso entre diferentes algoritmos, pues cada una estará rellena con una trama (de rayas negras) en un sentido diferente (horizontal, vertical, diagonales, etc).

Cuando se desee limpiar las marcas de las soluciones en el mapa, bastará con pulsar el botón *Limpiar Soluciones*.

B5. Funcionalidad como Simulador de mSS-HEXA

Como ya comentamos, mSS-Hexa es un simulador, aparte de un editor de mapas y visualizador de soluciones. Esto lo ponen de manifiesto varios aspectos ya descritos, como la definición de una serie de estructuras que representan entidades de la realidad, como pueden ser el terreno (con sus tipos, alturas, etc), las unidades militares, que cuentan con características comunes a la realidad (despliegues, recursos, armas, etc) y varias restricciones o condiciones como los consumos de salud (bajas de no combate) y recursos (gasto de combustible) asociados a las celdas.

Los consumos por defecto definidos para la simulación (según criterios propuestos por personal del ejército que se consultó), se pueden ver en la Tabla B1:

Pero además de estos factores, hablamos de mSS-Hexa como simulador porque implementa varias funciones que tienen su reflejo en la realidad (al igual que lo hacen muchos otros) y que hacen más realista el tratamiento del problema y sus soluciones.

	SALUD	*RECURSOS*
Normal	0,1	1
Agua	0,5	5
Bosque	0,3	3
Obstáculo	0	0

Tabla B1: Consumos por defecto para salud y recursos según el tipo de celda.

Estas funciones no se aplican en una simulación 'en tiempo real' como podría pensarse, sino que se usan a la hora de buscar las soluciones (tanto durante la construcción, como al asignar un coste a la solución) de modo si la unidad pasase a seguir el camino marcado por una de esas soluciones dentro de un simulador dinámico, los resultados se asemejarían mucho a los estimados en los costes de las soluciones que obtienen nuestros algoritmos.

Entre las funcionalidades implementadas se encuentran:

- *Distancia en número de celdas*: en lugar de obtener la distancia entre dos celdas con una fórmula matemática, ésta se calcula trazando una línea entre las celdas y contando el número de ellas que habría que atravesar para llegar de una a otra, lo cual es más realista, puesto que realmente para llegar de una celda a otra se dan un número entero de saltos, no un número real como obtendríamos con otras fórmulas.

- *Obstáculos Naturales*: se considera que hay un obstáculo natural, para la unidad, entre dos celdas si la diferencia de alturas es mayor que la que dicha unidad es capaz de superar. Por defecto se considera que este límite es 2, por lo que se considerará que hay un obstáculo natural entre celdas entre las que haya una diferencia de alturas mayor de 2.

- *Capacidad de Adquisición*: máxima distancia a la que pueden 'ver' (localizar a otras) las unidades, ya sean enemigas o la unidad del problema. Siguiendo los datos reales, esta distancia se fija en 9 Km, lo que correspondería a unas 18 celdas (considerando que la longitud de una celda (2*apotema) es 500 m).

- *Línea de Visión*: para determinar la visibilidad entre 2 celdas, se tiene esta función que comprueba dicha línea. Se considera que esta línea está obstruida (cada celda es oculta para la otra) si:

- Hay una celda de tipo bosque en la línea que las une (incluyéndolas a ellas) con una altura igual o superior a la más alta de las dos.
- Ambas celdas tienen la misma altura y existe una celda entre ellas con una altura 2 niveles superior.
- Hay una celda que corta una línea trazada desde la parte superior de una a la parte superior de otra (considerándolas en vista lateral de perfiles). Si es de tipo bosque, se considera que la celda intermedia es un nivel más alta (por la vegetación).

Además, esta línea está limitada por la capacidad de adquisición de la unidad que 'mira'.

- *Consumo de salud (energía)*: se considera como consumo de salud al atravesar una celda a la suma de la penalización que tiene asignada la celda (por su tipo y altura) y la letalidad de la misma.
- *Consumo de recursos*: se considera como consumo de recursos al atravesar una celda j viniendo de otra i a la suma de recursos por cambio de altura:
  - ninguno si la altura(i) = altura(j)
  - un valor mayor c a mayor cambio de altura, que será $c/2$ si $altura(i) > altura(j)$

  más el coste en recursos que tenga la celda j por defecto.

Referencias Bibliográficas

[1] Z. Michalewicz and D. B. Fogel. *How to Solve It: Modern Heuristics*. Springer Verlag, Berlín (Alemania), 2000.

[2] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[3] Eric Bonabeau, Marco Dorigo, and Guy Theraulaz. *Swarm intelligence: from natural to artificial systems*. Oxford University Press, Inc., New York, NY, USA, 1999.

[4] Zbigniew Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer, 1992.

[5] M. Dorigo and T. Stützle. The ant colony optimization metaheuristic: Algorithms, applications, and advances. In G.A. Kochenberger F. Glover, editor, *Handbook of Metaheuristics*, pages 251–285. Kluwer, 2002.

[6] A. Osyczka. Multicriteria optimization for engineering design. *In John S. Gero, editor, Design Optimization, pp.193-227. Academic Press*, 1985.

[7] Pierre-Paul Grassé. *Termitologia. Vol. I: Anatomie Physilogie Reproduction*, volume 1. Masson, Paris, 1982.

[8] Pierre-Paul Grassé. *Termitologia. Vol. II: Fondation des Sociétés Construction*, volume 2. Masson, Paris, 1984.

[9] Pierre-Paul Grassé. *Termitologia. Vol. III: Comportement Socialité Écologie Évolution Systématique*, volume 3. Masson, Paris, 1986.

[10] J.L.Denebourg, J.M.Pasteels, and J.C.Verhaeghe. Probabilistic behaviour in ants: a strategy of errors? *J. Theor. Biol.*, 105:259–271, 1983.

[11] J.L.Denebourg and S.Goss. Collective patterns and decision-making. *Ethology, Ecology & Evolution*, 1:295–311, 1989.

[12] S.Goss, R.Beckers, J.L.Denebourg, S.Aron, and J.M.Pasteels. How trail laying and trail following can solve foraging problems for ant colonies. In R.N.Hughes, editor, *Behavioural Mechanisms of Food Selection*, volume G. 20 of *NATO ASI Series*. Springer, Berlin, 1990.

[13] Pierre-Paul Grassé. La reconstruction du nid et les coordinations interindividuelles chez bellicositermes natalensis et cubitermes sp. la theorie de la stigmerie. *Insects Soc*, 6:41–80, 1959.

[14] M. Dorigo and G. Di Caro. The ant colony optimization meta-heuristic. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*, pages 11–32. McGraw-Hill, 1999.

[15] M. Dorigo, V. Maniezzo, and A. Colorni. The ant system: Optimization by a colony of cooperating agents. In *IEEE Transactions on Systems, Man, and Cybernetics - Part B*, volume 1, pages 29–41. 1996.

[16] M. Dorigo and L. M. Gambardella. Ant colony system: A cooperative learning approach to the traveling salesman problem. In *IEEE Transactions on Evolutionary Computation*, volume 1, pages 53–66. 1997.

[17] Boris V. Cherkassky, Andrew V. Goldberg, and Tomasz Radzik. Shortest paths algorithms: Theory and experimental evaluation. *Mathematical Programming*, 73:129–174, 1996.

[18] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys, editors. *The Traveling Salesman Problem*. John Wiley & Sons Ltd., 1985.

[19] G. B. Dantzig and J. H. Ramser. The truck dispatching problem. *Management Science*, 6(1):80–91, 1959.

[20] L. Bodin, B. Golden, A. Assad, and M. Ball. Routing and scheduling of vehicles and crews - the state of the art. volume 10, pages 63–212. Computers and Operations Research, 1983.

[21] Richard Bellman. On a routing problem. *Quarterly of Applied Mathematics*, 1(16):87–90, 1958.

[22] Edsger. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[23] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on*, 4(2):100–107, 1968.

[24] J. L. Bentley. Fast algorithms for geometric travelling salesman problem. *ORSA Journal on Computing*, 4(4):387–411, 1992.

[25] G. Reinelt. *The traveling salesman: Computational solutions for TSP applications*. Springer Verlag, 1994. LNCS 840.

[26] Jean-Yves Potvin. Genetic algorithms for the traveling salesman problem. *Annals of Operations Research*, 63(3):337–370, June 1996.

[27] H. Keko, M. Skok, and D. Skrlec. Artificial immune systems in solving routing problems. In *EUROCON 2003. Computer as a Tool. The IEEE Region 8*, volume 1, pages 62–66, 2003.

[28] *Bio-Inspired Algorithms for the Vehicle Routing Problem (Studies in Computational Intelligence)*. Springer, October 2008.

[29] Sam R. Thangiah, Olena Shmygelska, and William Mennell. An agent architecture for vehicle routing problems. In *SAC '01: Proceedings of the 2001 ACM symposium on Applied computing*, pages 517–521, New York, NY, USA, 2001. ACM.

[30] M. Dorigo, A. Colorni, and V. Maniezzo. Positive feedback as a search strategy. Technical Report 91-016, Dipartimento di Elettronica, Politecnico di Milano, Milan, Italy, 1991.

[31] M. Dorigo L. M. Gambardella. Ant-q: A reinforcement learning approach to the traveling salesman problem. In *Proceedings of the Twelfth International Conference on Machine Learning (ML-95)*, pages 252–260, Tahoe City, CA, USA, 1995.

[32] T. Stutzle and H. Hoos. Max-min ant system and local search for the traveling salesman problem. In *Evolutionary Computation, 1997., IEEE International Conference on*, pages 309–314, 1997.

[33] Feng Zuhong. An Improved Ant Colony Algorithm to Solve TSP Problem. *Journal of Northwest Minorities University (Natural Science)*, 22(2), June 2001.

[34] Jun Ouyang and Gui-Rong Yan. A multi-group ant colony system algorithm for tsp. volume 1, pages 117–121 vol.1, 2004.

[35] J. E. Bell and P. R. Mcmullen. Ant colony optimization techniques for the vehicle routing problem. *Adv. Eng. Inf.*, 18(1):41–48, 2004.

[36] Hui Fan, Zhen Hua, Jin-Jiang Li, and Da Yuan. Solving a shortest path problem by ant algorithm. In *Machine Learning and Cybernetics, 2004. Proceedings of 2004 International Conference on*, volume 5, pages 3174–3177, 2004.

[37] Yong Jiang, Wan-Liang Wang, and Yan-Wei Zhao. Solving the shortest path problem in vehicle navigation system by ant colony algorithm. In *ISCGAV'07: Proceedings of the 7th WSEAS International Conference on Signal Processing, Computational Geometry & Artificial Vision*, pages 188–192, Stevens Point, Wisconsin, USA, 2007. World Scientific and Engineering Academy and Society (WSEAS).

[38] Bradley S. Stewart and Chelsea C. White, III. Multiobjective A*. *J. ACM*, 38(4):775–814, 1991.

[39] Ronald Prescott Loui. Optimal paths in graphs with stochastic or multidimensional weights. *Communications of the ACM*, 26(9):670–676, 1983.

[40] Martin Desrochers, Jacques Desrosiers, and Marius Solomon. A new optimization algorithm for the vehicle routing problem with time windows. *Oper. Res.*, 40(2):342–354, 1992.

[41] M. M. Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Oper. Res.*, 35(2):254–265, 1987.

[42] L. Gambardella, E. Taillard, and G. Agazzi. Macs-vrptw: A multiple ant colony system for vehicle routing problems with time windows. In F. Glover D. Corne, M. Dorigo, editor, *New Ideas in Optimization*, pages 73–76. McGraw-Hill, 1999.

[43] B. Barán and M. Schaerer. A multiobjective ant colony system for vehicle routing problem with time windows. In *IASTED International Multi-Conference on Applied Informatics*, number 21 in IASTED IMCAI, pages 97–102, 2003.

[44] Alberto V. Donati, Luca M. Gambardella, Norman Casagrande, Roberto Montemanni, and Andrea E. Rizzoli. Time dependent vehicle routing problem with an ant colony system. Technical report, IDSIA report, 2003.

[45] Tong Zhen, Qiuwen Zhang, Wenshuai Zhang, and Zhi Ma. Hybrid ant colony algorithm for the vehicle routing with time windows. In *CCCM '08: Proceedings of the 2008 ISECS International Colloquium on Computing, Communication, Control, and Management*, pages 8–12, Washington, DC, USA, 2008. IEEE Computer Society.

[46] Carlos García-Martínez, Oscar Cordón, and Francisco Herrera. A new moea for multi-objective tsp and its convergence property analysis. In *Evolutionary Multi-Criterion Optimization*, number 2632 in LNCS, pages 342–354. Springer, 2003.

[47] Huang, Bo, Yao, Li, Raguraman, and K. Bi-level ga and gis for multi-objective tsp route planning. *Transportation Planning and Technology*, 29(2):105–124, April 2006.

[48] István Borgulya. An ec-memory based method for the multi-objective tsp. In *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 903–903, New York, NY, USA, 2007. ACM.

[49] F. Herrera C. García-Martínez, O. Cordón. A taxonomy and an empirical analysis of multiple objective ant colony optimization algorithms for the bi-criteria tsp. Technical Report SCI2S-2004-12, Research Group on Soft Computing and Intelligent Information Systems, University of Granada, España, Julio 2004.

[50] K. Wakuta. A multi-objective shortest path problem. *Mathematical methods of operations research*, 54:445–454, 2001.

[51] Andrea Raith and Matthias Ehrgott. A comparison of solution strategies for biobjective shortest path problems. *Computers and Operations Research*, 36(4):1299–1331, 2009.

[52] P. S. Maloney. An application of neural net technology to surveillance information correlation and battle outcome prediction. In *National Aerospace and Electronics Conference, NAECON 1989*, volume 2, pages 948–955. IEEE, 1989.

[53] D. Montana, G. Bidwell, G. Vidaver, and J. Herrero. Scheduling and route selection for military land moves using genetic algorithms. In P. Angeline, Z. Michalewicz, M. Schoenauer, X. Yao, and A. Zalzala, editors, *Congress on Evolutionary Computation*, pages 1118–1123. IEEE Press, 1999.

[54] I. Akgün and B. Ç. Tansel. Optimization of transportation requirements in the deployment of military units. *Computers & Operations Research*, 34(4):1158–1176, 2005.

[55] Tiao-ping Fu, Yu-shu Liu, and Jian-hua Chen. Improved genetic and ant colony optimization algorithm for regional air defense wta problem. In *ICICIC '06: Proceedings of the First International Conference on Innovative Computing, Information and Control*, pages 226–229, Washington, DC, USA, 2006. IEEE Computer Society.

[56] Gao Shang. Solving weapon-target assignment problems by a new ant colony algorithm. *Computational Intelligence and Design, International Symposium on*, 1:221–224, 2008.

[57] Ejército de Tierra Español en. http://es.wikipedia.org/wiki/Ejército_de_Tierra_Español.

[58] M.H. Karwan K. Thyagarajan, R. Batta and R.J. Szczerba. Planning dissimilar paths for military units. *Military Operations Research Journal*, 10(1):25–42, 2005.

[59] W. Matthew Carlyle, Johannes O. Royset, and R. Kevin Wood. Routing military aircraft with a constrained shortest-path algorithm, 2007.

[60] S. Alonso, O. Cordón, I. Fernández de Viana, and F. Herrera. La metaheurística de optimización basada en colonias de hormigas: Modelos y nuevos enfoques. In G. Joya, M.A. Atencia, A. Ochoa, and S. Allende, editors, *Optimización Inteligente: Técnicas de Inteligencia Computacional para Optimización*, pages 261–313. Servicio de Publicaciones de la Universidad de Málaga, 2004.

[61] M. Dorigo, G. Di Caro, and L.M. Gambardella. Ant algorithms for discrete optimization. *Artificial Life*, 5(2):137–172, 1999.

[62] T. Stützle and H. H. Hoos. The MAX-MIN ant system and local search for the travelling salesman problem. In T. Bäck, Z. Michalewicz, and X. Yao, editors, *Proceedings of the 1997 IEEE International Conference on Evolutionary Computation (ICEC'97)*, pages 309–314. IEEE Press, Piscataway, NJ, 1997.

[63] T. Stützle and H. H. Hoos. Max-min ant system. In *Future Generation Computer Systems*, volume 8, pages 889–914. 2000.

[64] B. Bullnheimer, R. F. Hartl, and C. Strauss. A new rank-based version of the ant system: A computational study. *Central European Journal for Operations Research and Economics*, 1(7):25–38, 1999.

[65] V. Maniezzo and A. Colorni. The ant system applied to the quadratic assignment problemm. In *IEEE Transactions on Data and Knowledge Engineering*, volume 5, pages 769–778. 1999.

[66] A. Colorni, M. Dorigo, V. Maniezzo, and M. Trubian. Ant system for job-shop scheduling. *JORBEL - Belgian Journal of Operations Research, Statistics and Computer Science*, 1(34):39–53, 1994.

[67] R. Schoonderwoerd, O. Holland, J. Bruten, and L. Rothkrantz. Ant-based load balancing intelecommunications networks. *Adaptative Behavior*, 2(5):169–207, 1996.

[68] G. Di Caro and M. Dorigo. Antnet: Distributed stigmergetic control for communications networks. *Journal of Artificial Intelligence Research*, (9):317–365, 1998.

[69] B. Bullnheimer, R. F. Hartl, and C. Strauss. An improved ant system algorithm for the vehicle routing problem. *Annals of Operations Research*, (89):319–328, 1999.

[70] L. M. Gambardella and M. Dorigo. Ant colony system hybridized with a new local search for the sequential ordering problem. *INFORMS Journal on Computing*, 3(12):237–255, 2000.

[71] T. Stützle. An ant approach to the flow shop problem. In *Proceedings of the 6th European Congress on Intelligent Techniques & Soft Computing (EUFIT'98)*, volume 3, pages 1560–1564. Verlag Mainz, 1998.

[72] D. Costa and A. Hertz. Ant can colour graphs. *Journal of the Operational Research Society*, (48):295–305, 1997.

[73] R. S. Parpinelli, H. S. Lopes, and A. A. Freitas. Data mining with an ant colony optimization algorithm. In *IEEE Transaction on Evolutionary Computation*, volume 6, pages 321–332. 2002.

[74] J. Casillas, O. Cordón, and F. Herrera. Learning fuzzy rules using ant colony optimization algorithms. In M. Dorigo, M. Middendorf, and T. Stützle, editors, *Abstract proceedings of ANTS2000 - From Ant Colonies to Artificial Ants: A Series of International Workshops on Ant Algorithms*, pages 13–21, 2000.

[75] L. M. de Campos, J. M. Fernández-Luna, J. A. Gámez, and J. M. Puerta. Ant colony optimization for learning bayesian networks. *International Journal of Approximate Reasoning*, 3(31):291–311, 2002.

[76] B. Mendoza and C.A. Coello. An approach based on the use of the ant system to design combinatorial logic circuits. *Mathware & Soft Computing*, 2-3(9):235–250, 2002.

[77] T. Bäck. Evolutionary algorithms in theory and practice. *Oxford University Press, New York*, 1996.

[78] F.Y. Edgeworth. Mathematical Physics. *P. Keagan, London, England*, 1881.

[79] V. Pareto. Cours d'economie politique. *volume I and II. F. Rouge, Lausanne*, 1896.

[80] L. Gacôgne. Multiple objective optimization of fuzzy rules for obstacles avoiding by an evolution algorithm with adaptative operators. In *Proceedings of the Fifth International Mendel Conference on Soft Computing (Mendel'99)*, pages 236–242, Brno, Czech Republic, Junio 1999.

[81] P. Grignon, J. Wodziack, and G. M. Fadel. Bi-objective optimization of components packing using a genetic algorithm. In *NASA/AIAA/ISSMO Multidisciplinary Design and Optimization Conference*, pages 352–362, Seattle, Washington, Septiembre 1996.

[82] D. Lee. Multiobjective design of a marine vehicle with aid of design knowledge. *International Journal for Numerical Methods in Engineering*, 40:2665–2677, 1997.

[83] C.A. Coello Coello, D.A. Van-Veldhuizen, and G.B. Lamont. Evolutionary algorithms for solving multi-objective problems. *Kluwer Academic Publishers, New York, ISBN 0-3064-6762-3*, 2002.

[84] C.M. Fonseca and P.J. Fleming. Multiobjetive Optimization and Multiple Constraint Handling with Evolutionary Algorithms - Part I: A Unified Formulation. *IEEE Transactions on Systems, Man and Cybernetics - Part A: Systems and Humans, 28(1):26-37*, 1998.

[85] D.A. Van-Veldhuizen. Multiobjective evolutionary algorithms: classifications, analyses and new innovations. *Ph.D. Thesis, AFIT/DS/ENG/99-01, Air Force Institute of Technology, Wright-Patterson AFB, Ohio*, 1999.

[86] J. Horn, N.Ñafpliotis, and D.E. Goldberg. A niched Pareto genetic algorithm for multiobjective optimization. *In Z. Michalewicz, editor, Proceedings of the 1st IEEE Conference on Evolutionary Computation, pp.82-87. IEEE Press, Piscataway, New Jersey*, 1994.

[87] J.B. Zydallis, D.A. Van-Veldhuizen, and G.B. Lamont. A statistical comparison of multiobjective evolutionary algorithms including the MOMGA-II. *In Zitzler, E., Deb, K., Thiele, L., Coello, C.A.C., and Corne, D., editors, 1st Intl. Conf. on Evolutionary Multi-Criterion Optimization, pp.226-240. Springer-Verlag. Lecture Notes in Computer Science, vol.1993*, 2001.

[88] M. Erickson, A. Mayer, and J. Horn. The Niched Pareto Genetic Algorithm 2 Applied to the Design of Groundwater Remediation Systems. *In Zitzler, E., Deb, K., Thiele, L., Coello, C.A.C., and Corne, D., editors, 1st Intl. Conf. on Evolutionary Multi-Criterion Optimization, pp.681-695. Springer-Verlag. Lecture Notes in Computer Science, vol.1993*, 2001.

[89] N. Srinivas and K. Deb. Multiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary Computation, 2(3):221-248*, 1994.

[90] K. Deb. An efficient constraint handling method for genetic algorithms. *Computer Methods in Applied Mechanics and Engineering, 186(2/4):311-338*, 2000.

[91] J. Knowles and D. Corne. Approximating the nondominated front using the pareto archived evolution strategy. *Evolutionary Computation, 8(2):149-172*, 2000.

[92] J. Knowles, R. Watson, and D. Corne. Reducing local optima in single objective problems by multi-objectivization. *In Zitzler, E., Deb, K., Thiele, L., Coello, C.A.C., and Corne, D., editors, 1st Intl. Conf. on Evolutionary Multi-Criterion Optimization, pp.268-282. Springer-Verlag. Lecture Notes in Computer Science, vol.1993*, 2001.

[93] E. Zitzler and L. Thiele. Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach. *IEEE Transactions on Evolutionary Computation, 3(4):257-271*, 1999.

[94] E. Zitzler, M. Laumanns, and L. Thiele. SPEA2: improving the Strength Pareto Evolutionary Algorithm. *Technical Report 103. Computer Engineering and Networks Laboratory (TIK), Swiss Federal Institute of Technology (ETH) Zurich*, 2001.

[95] F. de Toro, J. Ortega, J.Fernandez, and A.F Diaz. Parallel genetic algorithm for multiobjective optimization. *10th Euromicro Workshop on Parallel, Distributed and Network-based processing, IEEE Computer Society, pp. 384-391*, 2002.

[96] E. Morales C. E. Mariano. A multiple objective ant-q algorithm for the design of water distribution irrigation networks. Technical Report

HC-9904, Instituto Mexicano de Tecnología del Agua, Mexico, Junio 1999.

[97] Steffen Iredi, Daniel Merkle, and Martin Middendorf. Bi-criterion optimization with multi colony ant algorithms. In E. Zitzler, K. Deb, L. Thiele, C. A. Coello Coello, and D. Corne, editors, *Proceedings of the First International Conference on Evolutionary Multi-Criterion Optimization (EMO 2001)*, volume 1993 of *LNCS*, pages 359–372, Berlin, 2001. Springer.

[98] K. Doerner, W. J. Gutjahr, R. F. Hartl, C. Strauss, and C. Stummer. Pareto ant colony optimization: A metaheuristic approach to multiobjective portfolio selection. *Annals of Operations Research*, 131(1-4):79–99, 2004.

[99] A.C. Zecchin, A.R. Simpson, H.R. Maier, and J.B. Nixon. Parametric study for ant algorithm applied to water distribution system optimization. *IEEE Trans. Evol. Comput.,vol.9,2005, pp.175-191*, 1996.

[100] Xuyao Luo, Fang Yu, and Jun Zhang. Study of parametric relation in ant colony optimization approach to traveling salesman problem. In *ICIC 2006*, number 4115 in LNBI, pages 22–32. Springer-Verlag, 2006.