La Metaheurística de Optimización Basada en Colonias de Hormigas: Modelos y Nuevos Enfoques*

Sergio Alonso, Oscar Cordón, Iñaki Fernández de Viana, Francisco Herrera

Departamento de Ciencias de la Computación e Inteligencia Artificial, E.T.S. Ingeniería Informática, C/ Periodista Daniel Saucedo Aranda s/n, 18071 Granada (España)
zerjio@zerjio.com, ocordon@decsai.ugr.es, ijfviana@ugr.es, herrera@decsai.ugr.es

Abstract. En este trabajo se ofrece una perspectiva general de la metaheurística de Optimización Basada en Colonias de Hormigas, describiendo varios de los modelos algorítmicos existentes, y poniendo un especial énfasis en el Sistema de la Mejor-Peor Hormiga y en algunas de las nuevas tendencias existentes en el campo como, por ejemplo, la Paralelización de Colonias de Hormigas.

Palabras Clave. Optimización Basada en Colonias de Hormigas, Sistema de la Mejor-Peor Hormiga, Paralelización de Colonias de Hormigas, Viajante de Comercio

1 Introducción

Existen problemas de optimización combinatoria complejos en diversos campos como la economía, el comercio, la ingeniería, la industria o la medicina. Sin embargo, a menudo estos problemas son muy difíciles de resolver en la práctica. El estudio de esta dificultad inherente para resolver dichos problemas tiene cabida en el campo de la teoría de las Ciencias de la Computación, ya que muchos de ellos pertenecen a la clase de problemas NP-duros, lo que significa que no existe un algoritmo conocido que los resuelva en un tiempo polinomial [47].

Día tras día, siguen apareciendo nuevos problemas de este tipo, lo que ha dado lugar a que se hayan realizado muchas propuestas de algoritmos para tratar de solucionarlos. Las técnicas existentes se pueden clasificar básicamente en algoritmos exactos o aproximados. Los algoritmos exactos intentan encontrar una solución óptima y demostrar que la solución obtenida es de hecho la óptima global; estos algoritmos incluyen técnicas como, por ejemplo: procesos de vuelta atrás (*backtracking*), Ramificación y poda (*branch and bound*), programación dinámica, etc. [84, 13]. Debido a que los algoritmos exactos muestran un rendimiento pobre para muchos problemas se han desarrollado múltiples tipos de algoritmos aproximados que proporcionan soluciones de alta calidad para estos problemas

* Trabajo realizado en el marco del proyecto "Mejora de Metaheurísticas mediante Hibridación y sus Aplicaciones" de la Universidad de Granada.

combinatorios (aunque no necesariamente la óptima) en un tiempo computacional breve.

Los algoritmos aproximados se pueden clasificar en dos tipos principales: *algoritmos constructivos* y *algoritmos de búsqueda local*. Los primeros se basan en generar soluciones desde cero añadiendo componentes a cada solución paso a paso. Un ejemplo bien conocido son las *heurísticas de construcción voraz* o heurísticas *greedy* [13]. Su gran ventaja es la velocidad: normalmente son muy rápidas y, además, a menudo devuelven soluciones razonablemente buenas. Sin embargo, no puede garantizarse que dichas soluciones sean óptimas con respecto a pequeños cambios a nivel local. En consecuencia, una mejora típica es refinar la solución obtenida por la heurística voraz utilizando una búsqueda local. Los algoritmos de búsqueda local intentan repetidamente mejorar la solución actual con movimientos a soluciones vecinas (con la esperanza de que sean mejores). El caso más simple son los algoritmos de mejora iterativos: si en el vecindario de la solución actual s se encuentra una solución mejor s', ésta reemplaza la solución actual y se continua la búsqueda a partir de s'; si no se encuentra una solución mejor en el vecindario, el algoritmo termina en un óptimo local.

Desafortunadamente, los algoritmos de mejora iterativos pueden estancarse en soluciones de baja calidad (óptimos locales muy lejanos al óptimo global). Para permitir una mejora adicional en la calidad de las soluciones, la investigación en este campo en las últimas dos décadas ha centrado su atención en el diseño de técnicas de propósito general para guiar la construcción de soluciones o la búsqueda local en las distintas heurísticas. Estas técnicas se llaman comúnmente *metaheurísticas* [83, 48, 104] y consisten en conceptos generales empleados para definir métodos heurísticos. Dicho de otra manera, una metaheurística puede verse como un marco de trabajo general referido a algoritmos que puede aplicarse a diversos problemas de optimización (combinatoria) con pocos cambios significativos si ya existe previamente algún método heurístico específico para el problema. De hecho, las metaheurísticas son ampliamente reconocidas como una de las mejores aproximaciones para atacar los problemas de optimización combinatoria [2, 76, 88].

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. Algunos ejemplos de metaheurísticas son: *Enfriamiento simulado* [1, 64], *búsqueda tabú* [49], *búsqueda local iterativa* (*"iterated local search"*)[66], *algoritmos de búsqueda local con vecindario variable* (*"variable neighborhood search"*)[57], GRASP (*"greedy randomized adaptative search procedures"*) [39, 40] y *algoritmos evolutivos* [5, 6, 60]. Una metaheurística relativamente reciente es la *Optimización basada en Colonias de Hormigas* (OCH) (*"Ant Colony Optimization"*, ACO en inglés), la cual se inspira en el comportamiento que rige a las hormigas de diversas especies para encontrar los caminos más cortos entre las fuentes de comida y el hormiguero. De hecho, desde el trabajo inicial de Dorigo, Maniezzo y Colorni en el *Sistema de Hormigas* (SH) (*"Ant System"*, AS en inglés), la OCH se está convirtiendo en un campo de investigación importante: un gran número de autores han desarrollado modelos cada vez más sofisticados para

solucionar de manera satisfactoria un gran número de problemas de optimización combinatoria. Igualmente se ha incrementado el número de desarrollos teóricos sobre los algoritmos que se proponen.

Este trabajo repasa las bases de los algoritmos OCH, profundizando más concretamente en uno de los algoritmos existentes, el *Sistema de la Mejor-Peor Hormiga*; así como en las *estrategias de paralelización* que actualmente han sido desarrolladas o están siendo estudiadas.

La estructura del capítulo es la siguiente. En la *sección 2* se presentará el comportamiento real de las colonias de hormigas, a partir del cual se inspiraron para construir la OCH. A continuación, en la *sección 3* se describirá la transición desde las hormigas naturales hasta las hormigas artificiales, se resumirán las diferencias y similitudes entre ambas, se expondrá el modo de trabajo general de un algoritmo OCH y se discutirán, los tipos de problemas resolubles mediante OCH. En la *sección 4* se mostrarán algunos de los algoritmos OCH existentes, mientras que las distintas aplicaciones de los mismos se presentarán en la *sección 5*. En la *sección 6* describiremos la relación existente entre la OCH y otras metaheurísticas. Los aspectos teóricos de la OCH se reseñarán en la *sección 7*. Una vez presentadas todas las generalidades de la OCH pasaremos a analizar de manera mas concreta y exhaustiva el Sistema de la Mejor-Peor Hormiga (SMPH) en la *sección 8*, así como algunos resultados empíricos obtenidos por dicho algoritmo. La *sección 9* expondrá nuevas tendencias en la OCH, prestando especial atención a la paralelización de colonias de hormigas, y, finalmente, en la *sección 10* concluiremos el trabajo con unos comentarios finales.

2 Las Colonias de Hormigas Naturales

Las hormigas son insectos sociales que viven en colonias y que, debido a su colaboración mutua, son capaces de mostrar comportamientos complejos y realizar tareas difíciles desde el punto de vista de una hormiga individual. Un aspecto interesante del comportamiento de muchas especies de hormigas es su habilidad para encontrar los caminos más cortos entre su hormiguero y las fuentes de alimento. Este hecho es especialmente interesante si se tiene en cuenta que muchas de las especies de hormigas son casi ciegas, lo que evita el uso de pistas visuales.

Mientras que se mueven entre el hormiguero y la fuente de alimento, algunas especies de hormigas depositan una sustancia química denominada *feromona* (una sustancia que puede "olerse"). Si no se encuentra ningún rastro de feromona, las hormigas se mueven de manera básicamente aleatoria, pero cuando existe feromona depositada, tienen mayor tendencia a seguir el rastro. De hecho, los experimentos realizados por biólogos han demostrado que las hormigas prefieren de manera probabilística los caminos marcados con una concentración superior de feromona [87, 50]. En la práctica, la elección entre distintos caminos toma lugar cuando varios

caminos se cruzan. Entonces, las hormigas eligen el camino a seguir con una decisión probabilística sesgada por la cantidad de feromona: cuanto más fuerte es el rastro de feromona, mayor es la probabilidad de elegirlo. Puesto que las hormigas depositan feromona en el camino que siguen, este comportamiento lleva a un proceso de auto-refuerzo que concluye con la formación de rastros señalados por una concentración de feromona elevada. Este comportamiento permite además a las hormigas encontrar los caminos más cortos entre su hormiguero y la fuente del alimento [50].[1]

En la *Figura 1* se ilustra cómo este mecanismo permite a las hormigas encontrar el camino más corto. Inicialmente no existe ningún rastro de feromona en el medio y, cuando una hormiga llega a una intersección, elige de manera aleatoria una de las bifurcaciones posibles. Según transcurre el tiempo y mientras que las hormigas están recorriendo los caminos más prometedores, estos van recibiendo una cantidad superior de feromona. Esto ocurre gracias a que al ser los caminos más cortos, las hormigas que los siguen consiguen encontrar la comida más rápidamente, por lo que comienzan su viaje de retorno antes. Entonces, en el camino más corto habrá un rastro de feromona ligeramente superior y, por lo tanto, las decisiones de las siguientes hormigas estarán dirigidas en mayor medida a dicho camino. Además, este camino recibirá una proporción mayor de feromona por las hormigas que vuelven por él que por las que vuelven por el camino más largo. Este proceso finaliza haciendo que la probabilidad de que una hormiga escoja el camino más corto aumente progresivamente y que al final el recorrido de la colonia converja al más corto de todos los caminos posibles.

Esta convergencia se complementa con la acción del entorno natural que provoca que la feromona se evapore transcurrido un cierto tiempo. Así, los caminos menos prometedores pierden progresivamente feromona porque son visitados cada vez por menos hormigas. Sin embargo, algunos estudios biológicos han demostrado que los rastros de feromona son muy persistentes -la feromona puede permanecer desde unas pocas horas hasta varios meses dependiendo de varios aspectos distintos, como la especie de las hormigas, el tipo de suelo, ... [12]-, lo que provoca una menor influencia del efecto de la evaporación en el proceso de búsqueda del camino más corto.

En [12], numerosos experimentos muestran que, debido a la gran persistencia de feromona, es difícil que las hormigas “olviden” un camino que tiene un alto nivel de feromona aunque hayan encontrado un camino aún más corto. Hay que tener en cuenta que si se traslada este comportamiento directamente al ordenador para diseñar un algoritmo de búsqueda podemos encontrarnos con que se quede rápidamente estancado en un óptimo local. Volveremos sobre este punto más tarde en este trabajo (*sección 3.3*).

[1] Nótese que las hormigas solo se comunican de manera indirecta, a través de modificaciones del espacio físico que perciben. Esta forma de comunicación se denomina *Estimergia Artificial*.

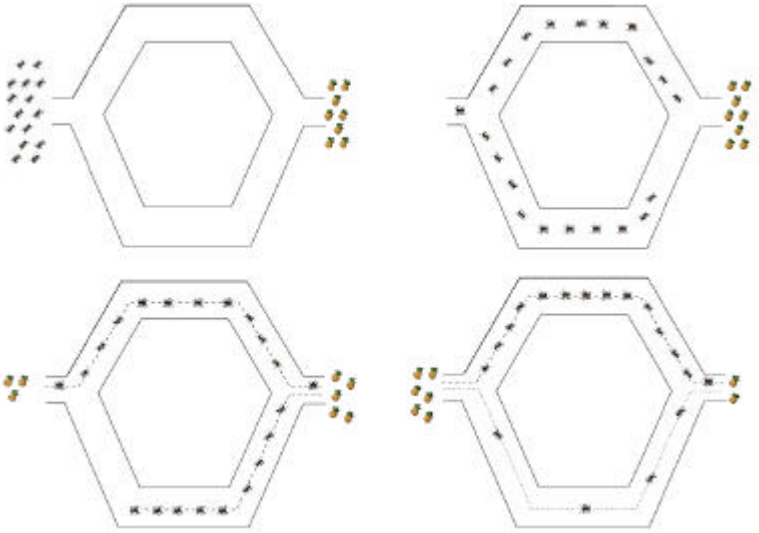

Fig. 1. *El comportamiento de la colonia termina por obtener el camino más corto entre dos puntos. Basado en una figura de [12]*

3 De las Hormigas Naturales a la Metaheurística de Optimización Basada en Colonias de Hormigas

Los algoritmos de OCH se inspiran directamente en el comportamiento de las colonias reales de hormigas para solucionar problemas de optimización combinatoria. Se basan en una colonia de hormigas artificiales, esto es, unos agentes computacionales simples que trabajan de manera cooperativa y se comunican mediante rastros de feromona artificiales.

Los algoritmos de OCH son esencialmente *algoritmos constructivos*: en cada iteración del algoritmo, cada hormiga construye una solución al problema recorriendo un grafo de construcción. Cada arista del grafo, que representa los posibles pasos que la hormiga puede dar, tiene asociada dos tipos de información que guían el movimiento de la hormiga:

- *Información heurística*, que mide la preferencia heurística de moverse desde el nodo *r* hasta el nodo *s*, o sea, de recorrer la arista a_{rs}. Se nota por h_{rs}. Las hormigas no modifican esta información durante la ejecución del algoritmo.
- *Información de los rastros de feromona artificiales*, que mide la "deseabilidad aprendida" del movimiento de *r* a *s*. Imita a la feromona real que depositan las hormigas naturales. Esta información se modifica durante la ejecución del algoritmo dependiendo de las soluciones encontradas por las hormigas. Se nota por t_{rs}.

En esta sección se presentan los pasos que llevan desde las hormigas reales a la OCH. A partir de ahora debe tenerse en cuenta que los algoritmos OCH presentan una doble perspectiva:

- Por un lado, son una abstracción de algunos patrones de comportamiento naturales relacionados con el comportamiento que permite encontrar el camino más corto.
- Por otro lado, incluyen algunas características que no tienen una contrapartida natural, pero que permiten que se desarrollen algoritmos para obtener buenas soluciones al problema que se pretende resolver (por ejemplo, el uso de información heurística que guíe el movimiento de las hormigas).

3.1 Tipos de Problemas Resolubles por la OCH

El tipo de problemas que se pueden resolver por medio de hormigas artificiales pertenece al grupo (restringido) de problemas de camino mínimo que se pueden caracterizar por los siguientes aspectos (seguimos principalmente la presentación expuesta en [30] y [36]):

- Existe un conjunto de restricciones Ω definido por el problema a solucionar.
- Existe un conjunto finito de componentes $N = \{n_1, n_2, ..., n_l\}$.
- El problema presenta diversos estados que se definen según secuencias ordenadas de componentes $d = <n_r, n_s, ..., u_n, ...>$ ($<r, s, \ldots, u, \ldots>$ para simplificar) sobre los elementos de N. Si Δ es el conjunto de todas las secuencias posibles, llamaremos $\tilde{\Delta}$ al conjunto de posibles (sub)secuencias que respetan las restricciones Ω. Los elementos en $\tilde{\Delta}$ definen los estados posibles. $|d|$ es la longitud de una secuencia d, esto es, el número de componentes en la secuencia.
- Existe una estructura de vecindario definida como sigue: d_2 es un vecino de d_1 si: (i) tanto d_1 como d_2 pertenecen a Δ, (ii) el estado d_2 puede alcanzarse desde d_1 en un paso lógico, es decir, si r es la última componente de la secuencia d_1, debe existir una componente $s \hat{I} Y$ tal que $d_2 = <d_1, s>$, o sea, debe existir una transición válida entre r y s. El vecindario alcanzable de d_1 es el conjunto que contiene todas las secuencias $d_2 \in \tilde{\Delta}$. Si $d_2 \notin \tilde{\Delta}$, diremos que d_2 está en el vecindario no alcanzable de d_1.
- Una solución S es un elemento de $\tilde{\Delta}$ que verifica todos los requisitos del problema.
- Existe un costo $C(S)$ asociado a cada solución S.
- En algunos casos, se puede asociar un costo o una estimación del mismo puede estar asociado a los estados.

Como hemos comentado todas las características anteriores se dan en problemas de optimización combinatoria que pueden representarse en forma de grafo ponderado $G = (N, A)$, donde A es el conjunto de aristas que conectan el conjunto de componentes N. El grafo G se denomina grafo de construcción G.[2] Por tanto, tenemos que:

- las componentes n_r son los nodos del grafo,
- los estados d (y por tanto las soluciones S) se corresponden con caminos en el grafo, esto es, secuencias de nodos o aristas,
- las aristas del grafo, a_{rs}, son conexiones/transiciones que definen la estructura del vecindario. $d_2 = <d_1,s>$ será vecino de d_1 si el nodo r es la última componente de d_1 y la arista a_{rs} existe en el grafo,
- deben existir los costos explícitos c_{rs} asociados con cada arista, y
- las componentes o las conexiones deben tener asociados rastros de feromona t, que representan un tipo de memoria indirecta y a largo plazo del proceso de búsqueda, como valores heurísticos h, que representan la información heurística disponible en el problema a resolver.

3.2 La Hormiga Artificial

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. Sin embargo, en algunos problemas, puede también construir soluciones no válidas que podrán ser penalizadas dependiendo de lo inadecuado de la solución. Tiene las siguientes propiedades [30, 36]:

- Busca soluciones válidas de costo mínimo para el problema a solucionar.
- Tiene una memoria L que almacena información sobre el camino seguido hasta el momento, esto es, L almacena la secuencia generada. Esta memoria puede usarse para (i) construir soluciones válidas, (ii) evaluar la solución generada, y (iii) reconstruir el camino que ha seguido la hormiga.
- Tiene un estado inicial $d_{inicial}$, que normalmente se corresponde con una secuencia unitaria y una o más condiciones t de parada asociadas.
- Comienza en el estado inicial y se mueve siguiendo estados válidos, construyendo la solución asociada incrementalmente.
- Cuando está en un estado $d_r = <d_{r-1}, r>$ (es decir, actualmente está localizada en el nodo r y ha seguido previamente la secuencia d_{r-1}, puede moverse a cualquier nodo s de su vecindario posible $N(r)$, definido como $N(r) = \{s \mid (a_{rs} \in A)\ y\ (<d_r, s> \in \tilde{\Delta})\}$.

[2] Como se dice en [31], el conjunto de aristas puede conectar completamente los componentes. En este caso, la implementación de las restricciones está completamente integrada en la política de construcción de las hormigas.

- El movimiento se lleva a cabo aplicando una regla de transición, que es función de los rastros de feromona que están disponibles localmente, de los valores heurísticos de la memoria privada de la hormiga y de las restricciones del problema.
- Cuando durante el procedimiento de construcción una hormiga se mueve desde el nodo *r* hasta el *s*, puede actualizar el rastro de feromona t_{rs} asociado al arco a_{rs}. Este proceso se llama *actualización en línea de los rastros de feromona paso a paso*.
- El procedimiento de construcción acaba cuando se satisface alguna condición de parada, normalmente cuando se alcanza un estado objetivo.
- Una vez que la hormiga ha construido la solución puede reconstruir el camino recorrido y actualizar los rastros de feromona de los arcos/componentes visitados/as utilizando un proceso llamado *actualización en línea a posteriori*. Este es el único mecanismo de comunicación entre las hormigas, utilizando la estructura de datos que almacena los niveles de cada arco/componente (memoria compartida).

3.3 Similitudes y Diferencias Entre las Hormigas Naturales y Artificiales

Las colonias de hormigas naturales y artificiales comparten una serie de características. Las más importantes pueden resumirse a continuación (véase [31] para una discusión más detallada):

- Uso de una colonia de individuos que interaccionan y colaboran para solucionar una tarea dada.
- Tanto las hormigas naturales como las artificiales modifican su "entorno" a través de una comunicación estimérgica basada en la feromona. En el caso de las hormigas artificiales, los rastros de feromona artificiales son valores numéricos que están disponibles únicamente de manera local.
- Ambas, las hormigas naturales y las artificiales, comparten una tarea común: la búsqueda del camino más corto (construcción iterativa del una solución de costo mínimo) desde un origen, el hormiguero (decisión inicial), hasta un estado final, la comida (última decisión).
- Las hormigas artificiales construyen las soluciones iterativamente aplicando una estrategia de transición local estocástica para moverse entre estados adyacentes, tal como hacen las hormigas naturales.

Sin embargo, estas características por sí solas no permiten desarrollar algoritmos eficientes para problemas combinatorios difíciles. De hecho, las hormigas artificiales viven en un mundo discreto y tienen algunas capacidades adicionales:

- Las hormigas artificiales pueden hacer uso de la información heurística (no solo de los rastros locales de feromona) en la política estocástica de transición que apliquen.
- Tienen una memoria que almacena el camino seguido por la hormiga.

- La cantidad de feromona depositada por la hormiga artificial es función de la calidad de la solución encontrada.[3] Otra gran diferencia se refiere al momento de depositar la feromona. Las hormigas artificiales normalmente sólo depositan feromona después de generar una solución completa.[4]
- Como se comentó en la *sección 2*, la evaporación de feromona en los algoritmos de OCH es diferente a como se presenta en la naturaleza, ya que la inclusión del mecanismo de evaporación es una cuestión fundamental para evitar que el algoritmo se quede estancado en óptimos locales. La evaporación de feromona permite a la colonia de hormigas artificiales olvidar lentamente su historia pasada para redireccionar su búsqueda hacia nuevas regiones del espacio. Esto evita una convergencia prematura del algoritmo hacia óptimos locales.
- Para mejorar la eficiencia y eficacia del sistema, los algoritmos OCH pueden enriquecerse con habilidades adicionales. Ejemplos típicos son la capacidad de mirar más allá de la siguiente transición (*"lookhead"*) [77], la *optimización local* [32, 97] y *"backtracking"* (cuyo uso no está muy extendido), que persiguen mejorar la eficacia, o la llamada *lista de candidatos* que contiene un conjunto de los estados vecinos mas prometedores [32, 30] para mejorar la eficiencia del algoritmo.

3.4 Modo de Funcionamiento y Estructura Genérica de un Algoritmo de OCH

Como se ha visto en las secciones anteriores, 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 estados adyacentes del problema (que puede representarse en forma de grafo con pesos). Este movimiento se realiza siguiendo una regla de transición que está basada en la información local disponible en las componentes (nodos). Esta información local incluye la información heurística y memorística (rastros de feromona) para guiar la búsqueda. Al moverse por el grafo de construcción, 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 se evalúa ésta y puede depositar una cantidad de feromona que es función de la calidad de su solución (*actualización en-línea a de los rastros 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 la hormigas

[3] Sin embargo, esta diferencia es relativa, ya que algunas especies de hormigas naturales depositan una cantidad superior de feromona cuando encuentran una fuente de alimento más rica [9].

[4] Aún así, como veremos a continuación, algunos algoritmos de OCH también modifican los rastros de feromona mientras que construyen la solución.

busquen y exploren nuevas regiones del espacio. Las acciones del demonio son acciones opcionales -que no tienen un contrapunto natural- para implementar tareas desde una perspectiva global que no pueden llevar a cabo las hormigas por la perspectiva local que ofrecen. Las capacidades adicionales que se mencionan en la *sección 3.3* se incluyen en estas acciones. Ejemplos son 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. En ambos casos, el demonio reemplaza la actualización en línea a posteriori de feromona y el proceso pasa a llamarse *actualización fuera de línea de rastros de feromona*.

La estructura de un algoritmo de OCH genérico es como sigue [30, 31]:

```
Procedimiento Metaheuristica_OCH()
  Inicializacion_de_parámetros
  mientras (criterio_de_terminación_no_satisfecho)
     Programación_de_actividades
        Generación_de_Hormigas_y_actividad()
        Evaporacion_de_Feromona()
        Acciones_del_demonio() {opcional}
     fin Programación_de_actividades
  fin mientras
fin Procedimiento

Procedimiento Generación_de_Hormigas_y_actividad()
  repetir en paralelo desde k=1 hasta m (numero_hormigas)
     Nueva_Hormiga(k)
  fin repetir en paralelo
fin Procedimiento

Procedimiento Nueva_Hormiga(id_Hormiga)
  inicializa_hormiga(id_Hormiga)
  L = actualiza_memoria_hormiga()
  mientras (estado_actual ≠ estado_objetivo)
     P = calcular_probabilidades_de_transición(A,L,Ω)
     siguiente_estado = aplicar_política_decisión(P,Ω)
     mover_al_siguiente_estado(siguiente_estado)
     si (actualizacion_feromona_en_linea_paso_a_paso)
        depositar_feromona_en_el_arco_vistado()
     fin si
     L = actualizar_estado_interno()
  fin mientras
  si (actualizacion_feromona_en_linea_a_posteriori)
     para cada arco visitado
        depositar_feromona_en_el_arco_visitado()
     fin para
  fin si
  liberar_recursos_hormiga(id_Hormiga)
fin Procedimiento
```

El primer paso incluye la inicialización del los valores de los parámetros que se tienen en consideración en el algoritmo. Entre otros, se deben fijar el rastro inicial de feromona asociado a cada transición, t_0, que es un valor positivo pequeño, normalmente el mismo para todas las componentes/conexiones, el número de hormigas en la colonia, *m*, y los pesos que definen la proporción en la que afectarán la información heurística y memorística en la regla de transición probabilística.[5]

El procedimiento principal de la metaheurística OCH controla, mediante el constructor `Programación_de_Actividades`, la planificación de las tres componentes mencionadas en esta sección: (i) la generación y puesta en funcionamiento de las hormigas artificiales, (ii) la evaporación de feromona, y (iii) las acciones del demonio. La implementación de este constructor determinará la sincronía existente entre cada una de las tres componentes. Mientras que la aplicación a problemas "clásicos" NP-duros (no distribuidos), normalmente usa una planificación secuencial, en problemas distribuidos como el enrutamiento en redes, el paralelismo puede ser explotado de manera sencilla y eficiente.

Como se ha comentado antes, varias componentes son o bien opcionales, como las acciones del demonio, o bien dependientes estrictamente del algoritmo de OCH especifico, por ejemplo 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 son mutuamente excluyentes y no suelen estar presentes a la vez ni faltar ambas al mismo tiempo (si las dos faltan, el demonio suele actualizar los rastros de feromona).

Por otro lado, hay que remarcar que el procedimiento `actualiza_memoria_hormiga()` se encarga de especificar el estado inicial desde el que la hormiga comienza su camino y, además almacenar la componente correspondiente en la memoria de la hormiga *L*. La decisión sobre cuál será dicho nodo depende del algoritmo específico (puede ser una elección aleatoria o una fija para toda la colonia, o una elección aleatoria o fija para cada hormiga, etc.).

Por último, comentar que los procedimientos `calcular_probabilidades_de_transición` y `aplicar_política_decisión` tienen en consideración el estado actual de la hormiga, los valores actuales de la feromona visibles en dicho nodo y las restricciones del problema Ω para establecer el proceso de transición probabilístico hacia otros estados válidos.

3.5 Relación entre la OCH y los Algoritmos de Hormigas

Es importante señalar que mediante el termino metaheurística OCH se entiende el modo genérico de operación de la OCH. El nombre algoritmo de OCH se usa para referirse a cada instancia específica del algoritmo genérico que se presentó en la

[5] Este aspecto se estudiará en mayor profundidad en la siguiente sección cuando se introduzcan algoritmos de OCH específicos.

sección 3.4, como por ejemplo los que se analizan en la *sección 4*. Además, la metaheurística OCH incluye un amplio rango de algoritmos que pueden tener formas muy variadas. Esto se debe principalmente a que existen distintos tipos de interacción complejos en el constructor `Programación_de_actividades` entre las actividades `Generación_de_Hormigas_y_actividad()`, `Evaporacion_de_Feromona()` y `Acciones_del_demonio()`. En cualquier caso, normalmente las hormigas se mueven de manera síncrona y la línea de acción general de los algoritmos de OCH actuales siguen un flujo de actividades simple [94, 95]. La razón principal para la gran generalidad de la heurística OCH es que fue definida a posteriori como un marco de trabajo general que engloba tanto aplicaciones existentes de problemas de optimización NP-duros como problemas inherentemente dinámicos como el enrutamiento en redes de telecomunicación (ver la *sección 5* para una breve descripción del mismo). Mientras que estos dos tipos de aplicaciones son similares desde una perspectiva global, debido al ámbito de aplicación tan distinto en ambas, los algoritmos y, en particular, las interacciones entre las tres actividades `Generación_de_Hormigas_y_actividad()`, `Evaporacion_de_Feromona()` y `Acciones_del_demonio()`son muy distintas desde una perspectiva local.

En cualquier caso, la metaheurística OCH no es lo suficientemente general como para cubrir la familia completa de algoritmos de hormigas, que pueden definirse (aunque no de manera muy precisa) como métodos aproximados para solucionar problemas combinatorios basados en características del comportamiento genérico de las hormigas naturales.[6] Algunos ejemplos de algoritmos de hormigas que no están incluidos en la metaheurística OCH son el *SH-rápido* (*"Fast Ant System"*) [102] y el *SH-Híbrido* (*"Hybrid Ant System"*) [46]. El primero es un algoritmo de construcción que se basa en el modo de operación de una única hormiga y que no utiliza de manera explicita la evaporación de feromona. El segundo consiste en un procedimiento de búsqueda local que hace uso de la información de los rastros de feromona para generar soluciones vecinas. En [100], se presenta un estudio de estos dos algoritmos para el TSP.[7]

3.6 Pasos a Seguir para Resolver un Problema Mediante OCH

Observando las aplicaciones actuales de la OCH, podemos identificar algunas directivas sobre como atacar problemas utilizando esta metaheurística. Estas directivas se pueden resumir en las seis tareas de diseño que se enumeran a continuación:

1. Representar el problema como un conjunto de componentes y transiciones o a través de un grafo ponderado (ver la *sección 3.1*) que será recorrido por las hormigas para construir soluciones.

---

[6] Todos los algoritmos OCH son algoritmos de hormigas mientras la aseveración contraria no se cumple.

[7] TSP, Traveling Salesman Problem o Problema del Viajante de Comercio, PVC.

2. Definir de manera apropiada el significado de los rastros de feromona t_{rs}, esto es, el tipo de decisión que inducen. Éste es un paso crucial en la implementación de un algoritmo de OCH y, a menudo, una buena definición de los rastros de feromona no es una tarea trivial. De hecho, normalmente implica tener un buen conocimiento del problema que se quiere solucionar.
3. Definir de manera apropiada la preferencia heurística de cada decisión que debe tomar una hormiga mientras que construye una solución, es decir, definir la información heurística h_{rs} asociada a cada componente o transición. Hay que remarcar que la información heurística es crucial para un buen rendimiento si no existen o no pueden ser aplicados algoritmos de búsqueda local.
4. Si es posible, implementar una búsqueda local eficiente para el problema que se desea solucionar, porque los resultados de muchas aplicaciones de la OCH a problemas de optimización combinatoria NP-duros demuestran que el mejor rendimiento se alcanza cuando se complementa con optimizaciones locales [30, 36].
5. Escoger un algoritmo de OCH especifico (algunos de los cuales serán descritos en la siguiente sección) y aplicarlo al problema que hay que solucionar, teniendo en cuenta, obviamente, todos los aspectos previamente comentados.
6. Refinar los parámetros del algoritmo de OCH. Un buen punto de partida para la especificación de los valores de los parámetros es usar configuraciones de parámetros que han demostrado ser buenas cuando se aplicaban en el algoritmo de OCH a problemas similares o a una gran variedad de problemas distintos. Otra posible alternativa para evitar el esfuerzo y tiempo necesarios para esta tarea es utilizar procedimientos automáticos de refinamiento de parámetros [9].

Debe quedar claro que los pasos arriba mostrados sólo dan una idea a grosso modo sobre la implementación del algoritmo de OCH. Además, en muchas ocasiones la implementación es un proceso iterativo, donde cuando se obtiene una visión más en profundidad del problema y del comportamiento del algoritmo, deben revisarse algunas de las decisiones iniciales. Por último, queremos insistir en el hecho de que probablemente los pasos más importantes son los cuatro primeros, ya que una elección poco acertada en esos puntos no suele poder arreglarse con un buen refinamiento de parámetros.

4 Modelos de Optimización Basada en Colonias de Hormigas

En la literatura se han propuesto diversos algoritmos que siguen la metaheurística OCH. Entre los algoritmos de OCH disponibles para problemas de optimización combinatoria NP-duros, se encuentran el *Sistema de Hormigas* (SH o *Ant System* (AS)) [33], el *Sistema de Colonia de Hormigas* (SCH o *Ant Colony System* (ACS)) [32], el *Sistema de Hormigas Max-Min* (SHMM, *Max-Min Ant System*) [98], el *SH con ordenación* (*Rank-Based Ant System*) [15] y el *Sistema de la Mejor-Peor Hormiga* (SMPH o *Best-Worst Ant System*) [24, 23]. En lo que sigue, presentaremos una pequeña descripción de estos algoritmos, salvo el SMPH, que será descrito con

más detalle y ofreciendo resultados empíricos en la *sección 8*.[8] Mientras que el SH presenta principalmente un interés histórico ya que fue el primer algoritmo de OCH, los otros cuatro normalmente alcanzan resultados mucho mejores. Una omisión importante en nuestra descripción de los algoritmos es el AntNet, un algoritmo de OCH que ha conseguido bastante éxito en el enrutamiento de paquetes en redes de comunicaciones. Aún así este algoritmo es más bien una aplicación específica que puede consultarse en [29].

Queremos reseñar que en lo sucesivo consideramos que tanto la feromona como la información heurística están asociadas únicamente a las conexiones, el cual es el caso de la gran mayoría de las aplicaciones de la OCH a problemas de secuenciación o asignación. Es más o menos directo extender la descripción al caso en el que la feromona esté asociada a componentes.

4.1 El Sistema de Hormigas

El SH [33], desarrollado por Dorigo, Maniezzo y Colorni en 1991, fue el primer algoritmo de OCH. Inicialmente, se presentaron 3 variantes distintas: *SH-densidad*, *SH-cantidad* y *SH-ciclo*, que se diferenciaban en la manera en que se actualizaban los rastros de feromona. En los dos primeros, las hormigas depositaban feromona mientras que construían sus soluciones (esto es, aplicaban una *actualización en-línea paso a paso de feromona*), con la diferencia de que la cantidad de feromona depositada en el SH-densidad es constante, mientras que la depositada en SH-cantidad dependía directamente de la deseabilidad heurística de la transición h_{rs}. Por último, en SH-ciclo, la deposición de feromona se lleva a cabo una vez que la solución está completa (actualización en línea a posteriori de feromona). Esta última variante era la que obtenía unos mejores resultados y es por tanto la que se conoce como SH en la literatura (y en el resto de este trabajo).

El SH se caracteriza por el hecho de que la actualización de feromona se realiza una vez que todas las hormigas han completado sus soluciones, y se lleva a cabo como sigue: primero, todos los rastros de feromona se reducen en un factor constante, implementándose de esta manera la evaporación de feromona. A continuación cada hormiga de la colonia deposita una cantidad de feromona que es función de la calidad de su solución. 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.

Las soluciones en el SH se construyen como sigue. En cada paso de construcción, una hormiga *k* escoge ir al siguiente nodo con una probabilidad que se calcula como:

---

[8] Para una descripción más detalla de los algoritmos, incluyendo comparaciones de su rendimiento cuando se aplican al problema del viajante de comercio, recomendamos [[30]], 26, 30, 85].

$$p_{rs}^{k} = \begin{cases} \dfrac{[t_{rs}]^{a} \cdot [h_{rs}]^{b}}{\sum_{u \in N_r^k} [t_{rs}]^{a} \cdot [h_{rs}]^{b}}, & si \quad s \in N_k(r) \\ 0, & en \quad otro \quad caso \end{cases} \qquad (1)$$

donde $N_k(r)$ es el vecindario alcanzable por la hormiga k cuando se encuentra en el nodo r, y a, b Î Â son dos parámetros que ponderan la importancia relativa de los rastros de feromona y la información heurística. Cada hormiga k almacena la secuencia que ha seguido hasta el momento y su memoria L_k, tal como se explicó antes, se utiliza para determinar $N_k(r)$ en cada paso de construcción.

Volviendo a los parámetros a y b, su función es la que sigue: si $a=0$, aquellos nodos con una preferencia heurística mejor tienen una mayor probabilidad de ser escogidos, haciendo el 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 $b=0$, 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 que el 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 dicho, la deposición de feromona se realiza una vez que todas las hormigas han acabado de construir sus soluciones. Primero, los rastros de feromona asociados a cada arco se evaporan reduciendo todos los rastros de feromona en un factor constante:

$$t_{rs} \leftarrow (1 - r) \cdot t_{rs} \qquad (2)$$

donde r Î$(0,1]$ es la tasa de evaporación. El siguiente paso de cada hormiga es recorrer de nuevo el camino que ha seguido (el camino esta almacenado en su memoria local L_k) y deposita una cantidad de feromona Δt_{rs}^{k} en cada conexión por la que ha viajado:

$$t_{rs} \leftarrow t_{rs} + \Delta t_{rs}^{k}, \quad \forall a_{rs} \in S_k \qquad (3)$$

donde $\Delta t_{rs}^{k} = f(C(S_k))$, es decir, la cantidad de feromona que se deposita depende de la calidad $C(S_k)$ de la solución S_k construida por la hormiga k.

Para resumir la descripción del SH, mostramos a continuación el procedimiento `Nueva_Hormiga` para este algoritmo de OCH en particular:

```
1 Procedimiento Nueva_Hormiga(id_Hormiga)
2  k = id_Hormiga; r = generar_estado_inicial; Sk = r
3  Lk = r
4  mientras (estado_actual ≠ estado_objetivo)
5     para cada sîNk(r) hacer  p_rs^k = [t_rs]^a · [h_rs]^b / Σ_{u∈N_r^k} [t_rs]^a · [h_rs]^b
6     siguiente_est=aplicar_política_de_decisión(P, Nk(r))
7     r = siguiente_est; Sk = <Sk, r>
8     ---
9     Lk=Lk∪r
10 fin mientras
            {se ejecuta el procedimiento
          Evaporación_de_feromona() se lanza y evapora la
          feromona en cada arco ars: τrs={1-ρ)·τrs}
11 para cada arco ars∈Sk hacer
12    τrs=τrs+f(c(Sk))
13 fin para
14 liberar_recursos_hormiga(id_Hormiga)
15 fin Procedimiento
```

La línea 8 vacía se incluye para hacer notar que no existe una actualización de feromona en-línea paso a paso y que, antes de la línea 12, el demonio debe haber aplicado la evaporación de feromona. De hecho, este es un ejemplo donde la construcción `Programación_de_actividades` interfiere con el funcionamiento de los procedimientos fundamentales de la metaheurística OCH, tal como se indicaba anteriormente.

Antes de finalizar esta sección es importante resaltar que los creadores del SH también propusieron una versión extendida del algoritmo que normalmente mejoraba los resultados obtenidos, llamada SH elitista [33]. En el SH elitista, 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 denomina 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:

$$t_{rs} = t_{rs} + e \cdot f(C(S_{mejor-global})), \quad \forall a_{rs} \in S_{mejor-global} \tag{4}$$

4.2 El Sistema de Colonias de Hormigas

El SCH [32] es uno de los primeros sucesores del SH que introduce tres modificaciones importantes con respecto a dicho algoritmo de OCH:

1. El SCH usa una regla de transición distinta, denominada regla proporcional pseudo-aleatoria. Sea k una hormiga situada en el nodo r, $q_0 \hat{I} [0,1]$ un parámetro y q un valor aleatorio en $[0,1]$, el siguiente nodo s se elige aleatoriamente mediante la siguiente distribución de probabilidad:

Si $q \leq q_0$:

$$p_{rs}^{k} = \begin{cases} 1, & si\ s = \arg \max_{u \in N_k(r)} \left\{ t_{ru} \cdot h_{ru}^{b} \right\} \\ 0, & en\ otro\ caso \end{cases} \quad (5)$$

si no ($q > q_0$):

$$p_{rs}^{k} = \begin{cases} \dfrac{[t_{rs}]^a \cdot [h_{rs}]^b}{\sum_{u \in N_r^k} [t_{rs}]^a \cdot [h_{rs}]^b}, & si \quad s \in N_k(r) \\ 0, & en\ otro\ caso \end{cases} \quad (6)$$

Como puede observarse, la regla tiene una doble intención: cuando q £ q_0, 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.

2. Sólo el demonio (y no las hormigas individualmente) actualiza la feromona, 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, $S_{mejor\text{-}global}$ (aunque en algunos trabajos iniciales se consideraba también una actualización basada en la mejor hormiga de la iteración [32], en OCH casi siempre se aplica la actualización por medio de la mejor global).

La actualización de la feromona se hace evaporando primero los rastros de feromona en todas las conexiones utilizadas por la mejor hormiga global (es importante recalcar que, en el SCH, la evaporación de feromona sólo se aplica a las conexiones de la solución, que es también la usada para depositar feromona) tal como sigue:

$$t_{rs} \leftarrow (1 - r) \cdot t_{rs}, \quad \forall a_{rs} \in S_{mejor-global} \quad (7)$$

A continuación, el demonio deposita feromona usando la regla:

$$t_{rs} \leftarrow t_{rs} + r \cdot f(C(S_{mejor-global})), \quad \forall a_{rs} \in S_{mejor-global} \quad (8)$$

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

3. 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:

$$t_{rs} \leftarrow (1-j)\cdot t_{rs} + j \cdot t_0 \qquad (9)$$

donde $j \hat{I} (0,1]$ es un segundo parámetro de decremento de feromona. Como puede verse, la regla de actualización en línea paso a paso incluye tanto la evaporación de feromona como la deposición de la misma. Ya que la cantidad de feromona depositada es muy pequeña (de hecho, t_0 es el valor del rastro de feromona inicial y se escogiese de tal manera que, en la práctica, se corresponda con el límite menor de rastro de feromona, esto es, con la elección de las reglas de actualización de feromona del SCH ningún rastro de feromona puede caer por debajo de t_0), la aplicación de esta regla hace que los rastros de feromona entre las conexiones recorridas por las hormigas disminuyan.[9] Así, esto lleva a una técnica de exploración adicional del SCH ya que las conexiones atravesadas por un gran número de hormigas son cada vez menos atractivas para el resto de hormigas que las recorren en la iteración actual, lo que ayuda claramente a que no todas las hormigas sigan el mismo camino.

Los procedimientos `Nueva_hormiga` y `Acciones_del_demonio` (que en este caso interactúa con el procedimiento de `evaporación_de_feromona`) para el SCH son como sigue:

```
1 Procedimiento Nueva_Hormiga(id_Hormiga)
2  k = id_Hormiga; r = Generar_Estado_Inicial; Sk = r
3  Lk = r
4  mientras (estado_actual ≠ estado_objetivo)
5     para cada s ∈ Nk(r) hacer  calcular brs = trs · hrs^b
6     q = generar_valor_aleatorio_en_[0,1]
      si (q <= q0)
         siguiente_estado = max(brs, Nk(r))
      si no
         para cada s ∈ Nk(r) hacer
                                  p_rs^k = b_rs / Σ_{u∈Nk(r)} b_ru
         sig_est=aplicar_política_de_decisión(P, Nk(r))
      fin si
```

[9] El SCH está de hecho basado en el Ant-Q, un algoritmo anterior propuesto por Gambardella y Dorigo [36]. Su única diferencia con respecto al SCH es la definición del término t_0 en la actualización en-línea paso a paso de feromona, que en Ant-Q es la evaluación adelantada del siguiente estado ($\gamma \cdot \max_{s \in Ni®}\{\tau_{rs}\}$). De cualquier manera, los resultados experimentales sugieren que el SCH está al mismo nivel de rendimiento y, por su mayor simplicidad es preferible al Ant-Q.

```
7     r = sig_est;  S_k =< S_k, r >
8     t_rs = (1 − j) · t_rs + j · t_0
9     L_k = L_k ∪ r
10 fin mientras
11 ---
12 ---
13 ---
14 liberar_recursos_hormiga(id_Hormiga)
15 fin Procedimiento

1 Procedimiento Acciones_del_demonio
2  para cada S_k hacer búsqueda_local(S_k) {opcional}
3  S_mejor-actual = mejor_solucion(S_k)
4  si (mejor(S_mejor-actual, S_mejor-global) )
5     S_mejor-global = S_mejor-actual
6  fin si
7  para cada arista a_rs ÎS_mejor-global hacer
                 {se      ejecuta      el      procedimiento
                 evaporación_de_feromona() se lanza y evapora
                 feromona en la arista a_rs : τ_rs = (1 - ρ)·τ_rs}
8     t_rs = t_rs + r¼f(C(S_mejor-global))
9  fin para
10 fin Procedimiento
```

4.3 El Sistema de Hormigas Max-Min

El SHMM [99, 94, 98], desarrollado por Stützle y Hoos en 1996, es una de las extensiones del SH que mejor rendimiento muestran. Extiende el SH en los siguientes aspectos:

1. Se aplica una actualización de los rastros de feromona fuera de línea, de manera similar a como se hace en el SCH. Después de que todas las hormigas hayan construido su solución cada rastro de feromona sufre una evaporación:

$$t_{rs} = (1 - r) \cdot t_{rs} \qquad \textbf{(10)}$$

y a continuación la feromona se deposita siguiendo la siguiente fórmula:

$$t_{rs} = t_{rs} + f(C(S_{mejor})), \quad \forall a_{rs} \in S_{mejor} \qquad \textbf{(11)}$$

La *mejor hormiga* a la que se le permite añadir feromona puede ser la que tiene una solución *mejor de la iteración* o la solución *mejor global*. Los resultados experimentales demuestran que el mejor rendimiento se obtiene incrementando gradualmente la frecuencia de escoger la mejor global para la actualización de feromona [94, 98].

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

2. Los valores posibles para los rastros de feromona están limitados al rango $[t_{min}, t_{max}]$. Por lo tanto, la probabilidad de un estancamiento del algoritmo disminuye al darle a cada conexión existente una probabilidad, aunque bastante pequeña, de ser escogida. En la práctica, existen heurísticas para fijar los valores de t_{min} y t_{max}. Se puede ver que, a causa de la evaporación de la feromona, el nivel máximo de feromona en los rastros está limitado a $t^*_{max} = 1/(r \times C(S^*))$, donde S^* es la solución óptima. Basándonos en este resultado, la mejor solución global puede usarse para estimar t_{max} sustituyendo S^* por $S_{mejor\text{-}global}$ en la ecuación de t^*_{max}. Para t_{min}, normalmente sólo es necesario escoger su valor de tal manera que sea un factor constante menor que t_{max} (ver [98] para más detalles sobre algunas de las distintas maneras de escoger t_{min}).

Para poder incrementar la exploración de nuevas soluciones, el SHMM utiliza en ocasiones re-inicializaciones de los rastros de feromona [97, 94, 98].

3. 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 dicha solución $S´$ en la ecuación de t^*_{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.

La estructura del procedimiento Acciones_del_demonio en el SHMM se muestra a continuación:

```
1 Procedimiento Acciones_del_demonio
2   para cada Sk hacer búsqueda_local(Sk)
3   Smejor-actual = mejor_solucion(Sk)
4   si (mejor(Smejor-actual,Smejor-global) )
5      Smejor-global = Smejor-actual
6   fin si
7   Smejor = decisión(Smejor-global,Smejor-actual)
8   para cada arista ars ∈ Smejor hacer
9      τrs = τrs + f(C(Smejor))
10     si (τrs < τmin)  τrs = τmin
11  fin para
12  si (condición_de_estancamiento)
13     para cada arista ars hacer τrs=τmax
14  fin si
15 fin Procedimiento
```

4.4 El Sistema de Hormigas con Ordenación

El *Sistema de Hormigas con Ordenación* ($SH_{ordenación}$) [15] 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, de nuevo, fuera de línea, tal como se muestra a continuación:

1. 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.

2. El demonio deposita feromona en las conexiones por las que han pasado las $s - 1$ mejores hormigas (hormigas elitistas). La cantidad de feromona depositada depende directamente del orden de la hormiga y de la calidad de su solución.

3. 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. Esta deposición de feromona se considera la más importante, de hecho, recibe un peso de *s*.

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:

$$t_{rs} \leftarrow t_{rs} + s \cdot \Delta t_{rs}^{mg} + \Delta t_{rs}^{orden} \qquad (12)$$

donde

$$\Delta t_{rs}^{mg} = \begin{cases} f(C(S_{mejor-global})), & si\ a_{rs} \in S_{mejor-global} \\ 0, & en\ otro\ caso \end{cases} \qquad (13)$$

$$\Delta t_{rs}^{orden} = \begin{cases} \sum_{m=1}^{s-1} (s-m) \cdot f(C(S'_m)), & si\ a_{rs} \in S'_m \\ 0, & en\ otro\ caso \end{cases} \qquad (14)$$

Finalmente, el procedimiento `Acciones_del_demonio` del $SH_{ordenación}$ presenta la siguiente estructura:

```
1 Procedimiento Acciones_del_demonio
2  para cada Sk hacer búsqueda_local(Sk) {opcional}
3  ordenar (S1, ..., Sm) en orden decreciente según la calidad
          de la solución: (S´1, ..., S´m)
4  si (mejor(S´1, Smejor-global) )
5     Smejor-global = S´1
6  fin si
7  desde μ = 1 hasta (σ - 1) hacer
```

```
8     para cada arista a_rs ∈ S´_μ hacer
9        τ_rs = τ_rs + (σ - μ)·f(C(S´_μ))
10    fin para
11 fin desde
12 para cada arista a_rs ∈ S_mejor-global hacer
13    τ_rs = τ_rs + σ · f(C(S´_μmejor-global))
14 fin para
15 fin Procedimiento
```

5. Aplicaciones de la Optimización Basada en Colonias de Hormigas

Los algoritmos de OCH se han aplicado a un gran número de problemas de optimización combinatoria diferentes. Las aplicaciones actuales de la OCH se distribuyen dentro de dos clases fundamentales. La primera clase de problemas está compuesta por los problemas de *optimización combinatoria NP-duros*, para los que las técnicas clásicas ofrecen a menudo un comportamiento pobre. 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 las hormigas. La segunda clase de aplicaciones se compone de *problemas dinámicos de caminos mínimos*, donde la instancia del problema que hay que solucionar cambia durante la ejecución del algoritmo. Estos cambios pueden afectar a la topología del problema, como por ejemplo la disponibilidad de los enlaces, etc. 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. Esta última clase incluye aplicaciones de la OCH al enrutamiento en redes de comunicaciones.

En vez de enumerar exhaustivamente las diversas aplicaciones de la OCH, vamos a describir brevemente la primera etapa histórica de desarrollo de aplicaciones. Para un repaso más profundo de las aplicaciones de la OCH, aconsejamos [30, 31, 35, 36].

El primer problema que fue atacado por un algoritmo de OCH fue el TSP, ya que este problema 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 casi inmediata. 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 [32, 98, 15, 23].

Cronológicamente, las dos aplicaciones siguientes fueron el problema de la asignación cuadrática (QAP) [69, 33] (los mejores algoritmos de OCH para este problema están descritos en [98, 68]) y el problema de la secuenciación de tareas (job-shop scheduling, JSP) [20] en 1994. Entre las aplicaciones posteriores se encuentran las primeras aplicaciones de enrutamiento en redes, comenzando en 1996 con el trabajo de Schoonderwoerd y otros [91] y el trabajo sobre AntNet por Di Caro y

Dorigo [29]. Ya en 1997, un año después de la publicación del primer artículo de revista sobre OCH en 1996 [33], el número de aplicaciones de la OCH comienza a incrementarse de manera considerable. Algunas aplicaciones de primeros de 1997 (aunque algunas de ellas aparecieron publicadas más tarde) incluyen problemas clásicos de enrutamiento de vehículos [14], de ordenación secuencial [44], de secuenciación (flow shop scheduling, FSS) [92], y de coloreo de grafos [26]. 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 y varios problemas de la mochila y de satisfacción de restricciones, entre otros. El lector interesado puede encontrar un resumen de las aplicaciones disponibles hacia finales del 2000 en [36]. Aparte de las aplicaciones anteriores, la OCH ha sido usada recientemente para aprendizaje automático *("machine learning"*), concretamente para el diseño de algoritmos de aprendizaje para estructuras de representación del conocimiento como las clásicas reglas lógicas [85, 86], reglas difusas [18, 3, 19] y redes bayesianas [28, 27], 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, 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, lo cual es remarcable, ya que muchos de esos problemas han atraído una gran cantidad de esfuerzo e investigación. Por otro lado, la metaheurística OCH está siendo aplicada a nuevos problemas reales con resultados prometedores (por ejemplo, la aplicación al diseño de circuitos lógicos combinatorios [71]).

6 Relación entre la OCH y otras Metaheurísticas

Los algoritmos de OCH tienen bastantes similitudes con otros algoritmos de optimización, y con aproximaciones al modelado y aprendizaje como por ejemplo la búsqueda heurística en grafos, los métodos de simulación de Monte Carlo y las redes neuronales. Estas similitudes se analizan en [31]. Además, la metaheurística OCH comparte bastantes aspectos comunes con otras metaheurísticas como la computación evolutiva y los algoritmos de estimación de distribuciones, por un lado, y con GRASP y la búsqueda local multi-arranque, por otro.

6.1 Relación entre OCH, Computación Evolutiva y Algoritmos de Estimación de Distribuciones

Como se señala en [31], existen similitudes entre el modo de operación de los algoritmos de OCH y los algoritmos evolutivos como el uso de una población de individuos que codifican soluciones al problema y que son generadas de manera estocástica.

Una gran diferencia es que, en la computación evolutiva, el conocimiento del problema está contenido en la población actual, mientras que en la OCH se almacena en la estructura memorística que guarda los rastros de feromona. Aún así, existe una clase específica de algoritmos evolutivos, llamados Algoritmos de Estimación de Distribuciones (EDAs) [65] que, tal como hacen los algoritmos de OCH, están basados en mantener una estructura memorística que representa una distribución de probabilidad definida sobre las variables del problema. Esta distribución de probabilidad se adapta durante la ejecución del algoritmo: se van generando soluciones según la distribución actual, que son utilizadas posteriormente para actualizar la distribución de probabilidad. Este proceso de repite iterativamente.

El algoritmo EDA mejor conocido es el Aprendizaje Incremental Basado en Poblaciones (*"Population-Based Incremental Learning"*) o PBIL [7], que es además el más parecido a los algoritmos de OCH [23], más concretamente al SCH. PBIL trabaja con un vector de probabilidades $P=(p_1, ..., p_n)$ de dimensión n igual al número de variables del problema, el cual codifica una distribución de probabilidad que representa un prototipo de buenas soluciones y que se usa para generar una población de posibles soluciones (vectores binarios) en cada iteración. Este vector de probabilidades sufre una adaptación durante la ejecución del algoritmo. En el modelo básico del PBIL, se actualiza dependiendo de la composición de la mejor solución generada en la iteración actual usando una regla de actualización similar a la regla de actualización de feromona fuera de línea del SCH. Por otro lado, las componentes de P también sufren mutaciones aleatorias para evitar la posibilidad de una convergencia prematura.

Las similitudes entre los algoritmos de OCH y PBIL fueron analizadas también por Monmarché y otros [80]. Estos autores presentaron un marco de trabajo común que llamaron Metaheurística de Búsqueda Probabilística (*"Probabilistic Search Metaheuristic"*) que incluye otro modelo EDA adicional, el Cruce Simulado de Bits (Bit-Simulated Crossover) [101]. Además, Dorigo y otros también han estudiado las similitudes entre la OCH y otras técnicas, y han identificado un gran grupo de algoritmos con características comunes que llamaron Búsqueda Basada en Modelos (Model-based Search) [106] (ver también la *sección 7*).

Por último, debemos recordar que las similitudes entre la OCH y la computación evolutiva - EDAs han motivado la integración de ciertos aspectos de los últimos en los algoritmos de OCH para mejorar su rendimiento. Este es el caso, en particular, del SMPH (ver la *sección 9*). De hecho, el SMPH incorpora varios conceptos de los EDAs, principalmente de PBIL, como la mutación de los rastros de feromona y la penalización (evaporación adicional de feromona) por la peor solución encontrada en la iteración actual.

6.2 Relación entre OCH, GRASP y Búsqueda Multi-arranque

Muchas metaheurísticas para problemas de optimización combinatoria NP-duros implementan alguna forma de búsqueda local multi-arranque, donde se generan

soluciones iniciales para la búsqueda local de manera iterativa. Estas metaheurísticas incluyen la búsqueda local iterativa ("*Iterated Local Search*") o ILS [66], la búsqueda de vecindario variable ("*variable neighborhood search*", VNS) [57], los algoritmos GRASP [39, 40] y los algoritmos meméticos ("*memetic algorithms*", AMs) [81]. De hecho, muchos algoritmos de OCH también pertenecen a esta clase de metaheurísticas.[10] En este caso, el modo de operación de los algoritmos de OCH se puede representar de la siguiente manera:

Mientras (NO se de la condición de parada) hacer

1. Construcción probabilística de soluciones por una colonia de hormigas.

2. Optimización local de esas soluciones.

3. Actualización de los rastros de feromona fuera de línea.

Una diferencia fundamental entre la OCH y la ILS, la VNS o los AMs es la manera en la que se eneran las soluciones. Mientras que la OCH construye soluciones partiendo desde cero, la ILS, la VNS y los AMs modifican soluciones existentes aplicando operadores apropiados que, cuando son utilizados sobre una o varias soluciones, devuelven una solución que representa una perturbación a la(s) solución(es) inicial(es). Adicionalmente, la OCH usa una forma explícita de memoria (en forma de rastros de feromona), lo que no es habitual en las otras tres metaheurísticas.

En todo caso, la OCH comparte la particularidad de la construcción de soluciones con GRASP, que construye soluciones probabilísticamente para posteriormente aplicarles una búsqueda local. En GRASP, este proceso en dos fases se repite numerosas veces y la mejor solución encontrada es la devuelta. En la fase de construcción de GRASP se genera un conjunto de candidatos en cada paso de construcción, ordenando primero las componentes de la solución según alguna información heurística e incluyendo posteriormente las componentes más prometedoras (componente voraz). A continuación, se realiza una decisión aleatoria entre las componentes del conjunto de candidatos, donde típicamente todos los miembros del conjunto de candidatos tienen la misma probabilidad de ser seleccionados (componente aleatoria). Una particularidad de GRASP es que la información heurística en cada paso tiene en cuenta la solución parcial que ya está disponible (componente adaptativa). De esta descripción, se desprende que los algoritmos GRASP y la OCH son distintos. Mientras que la OCH usa la información memorística en la historia del algoritmo (la cual está reflejada en los rastros de feromona), los algoritmos GRASP no consideran ese tipo de información para guiar el proceso de construcción. Por otro lado, la OCH no tiene por qué usar la información

[10] Hacer notar que esto es cierto para la mayoría de los algoritmos de OCH de mejor rendimiento cuando se aplican a problemas NP-duros, pero no se puede aplicar para cualquier algoritmo de OCH, por ejemplo para el enrutamiento en redes de telecomunicaciones.

heurística que es dependiente de la solución parcial, cosa que siempre realiza GRASP. Sin embargo, usar la información heurística adaptativa (dinámica) ha demostrado mejorar el rendimiento de la OCH para distintas aplicaciones [36].

7 Desarrollos Teóricos

Los resultados teóricos actuales sobre la OCH afectan principalmente a dos aspectos: (i) pruebas sobre la convergencia de los algoritmos de OCH y (ii) establecimiento de enlaces formalmente bien formados entre los algoritmos de OCH y otras técnicas algorítmicas relacionadas.

Stützle y Dorigo han demostrado las propiedades de convergencia para una clase de algoritmos OCH llamados OCHt_{min} [96]. Las características de los algoritmos de OCHt_{min} son que usan una estrategia de actualización de feromona elitista y límites inferiores en los valores de cualquier rastro de feromona. Estos autores probaron que, para cualquier cantidad pequeña $e > 0$ y para un número suficientemente grande del iteraciones t del algoritmo de OCHt_{min}, la probabilidad de encontrar al menos una solución óptima es $P^*(t) \geq 1 - e$ y que esa probabilidad tiende a 1 para $t \rightarrow \infty$. Lo más importante de este resultado es que se aplica a al menos dos de los algoritmos de OCH con más éxito (experimental): el SCH [32] y el SHMM [98].

Aún así, el primero en demostrar la convergencia de un algoritmo de OCH particular, el llamado Sistema de Hormigas Basado en Grafos (Graph-Based Ant System, GBAS), fue Gutjahr [54]. Demostró que (i) para cada $e > 0$, para un r fijo y para un número suficientemente grande de hormigas, la probabilidad P de que una hormiga fija construya la solución óptima en la iteración t es $P \geq 1 - e$ para todo $t \geq t_0$, con $t_0 = t_0(e)$; y (ii) para cada $e > 0$, para un número fijo de hormigas y para una tasa de evaporación r suficientemente cercana a 0, la probabilidad P de que una hormiga fija construya la solución óptima en la iteración t es $P \geq 1 - e$ para todo $t \geq t_0$, con $t_0 = t_0(e)$.

Aún así, el GBAS es un algoritmo de OCH que no ha sido aplicado a ningún problema de optimización combinatoria. De hecho, por la descripción del algoritmo y algunos cálculos sobre los parámetros para valores bajos de e que están implicados en el teorema, el algoritmo obtendría un progreso muy lento hacia las buenas soluciones y, por lo tanto, parece que no sería relevante para la OCH en la práctica.

En un trabajo reciente, Gutjahr [55] extiende los resultados de convergencia para el GBAS [54] a dos variantes del mismo, obteniendo el mismo tipo de resultados de convergencia que para el Enfriamiento Simulado [56]: convergencia de la solución actual hacia la solución óptima con probabilidad 1. Este resultado se establece para dos variantes del GBAS con (i) evaporación de feromona dependiente del tiempo y (ii) límites de feromona inferiores dependientes del tiempo.

Existen algunas diferencias entre las pruebas para el OCHt_{min} y el GBAS. La más importante tiene que ver con el tipo de convergencia probada. Mientras que para el OCHt_{min} se prueba la convergencia en valor (esto es, el algoritmo eventualmente encontrará la solución óptima), para el GBAS Gutjahr demuestra la convergencia en solución (es decir, el algoritmo convergerá a una situación en la que generará la solución óptima una y otra vez). Para propósitos prácticos es claro que es suficiente demostrar la convergencia en valor, porque básicamente todas las metaheurísticas devuelven la mejor solución encontrada durante el proceso de búsqueda y, por lo tanto, es suficiente generar la solución óptima al menos una vez.

Meuleau y Dorigo en [75] han establecido relaciones formales entre los algoritmos de OCH y el gradiente descendiente estocástico (Stochastic gradient descent, SGD), una técnica que ha sido ampliamente usada en aprendizaje automático [90, 79]. Muestran que una variante del SH puede ser interpretado como un algoritmo SGD que realiza un descenso de gradiente en el espacio de los rastros de feromona. Como se menciona en la *sección 6.1*, Dorigo y otros [37] dan una interpretación de la OCH como un método de búsqueda basado en modelos. Este tipo de técnicas se caracterizan por el hecho de que soluciones candidatas son generadas por un modelo probabilístico parametrizado que se actualiza después de cada iteración en función de las soluciones previamente vistas. Esta retroalimentación se usa para guiar la distribución de probabilidad para las siguientes iteraciones. En este marco de trabajo se establecen conexiones entre OCH y EDAs, SGD, el método de entropía cruzada y los algoritmos genéticos basados en modelos [82].

8 El Sistema de la Mejor - Peor Hormiga

El SMPH [23, 24], propuesto por Cordón y otros en 1999, es un algoritmo de OCH que incorpora conceptos de computación evolutiva.[11] Este algoritmo es otra extensión del SH. En esta sección vamos a examinarlo en detalle y a presentar algunos resultados empíricos obtenidos utilizándolo.

8.1 Estructura del SMPH

El SMPH utiliza la misma regla de transición de estados que el SH, así como la misma regla de evaporación de feromona que, al igual que en el $SH_{ordenación}$ y el SHMM, se aplica a todas las transiciones (ver la *sección 4.1* para más detalles). Además, tal como hace el SHMM, el SMPH siempre considera la explotación sistemática de optimizadores locales para mejorar las soluciones de las hormigas.

Formando el núcleo del SMPH podemos encontrar las siguientes tres acciones del demonio:

[11] La relación entre estas dos metaheurísticas ha sido analizada en la *sección 6.1*.

1. La *regla mejor-peor de actualización de rastros de feromona*, basada en la regla de actualización del vector de probabilidades de PBIL, refuerza las aristas que se encuentran en la mejor solución global. Además, penaliza cada conexión de la peor solución generada hasta el momento, $S_{peor\text{-}actual}$, que no se encuentre en la mejor global realizando una evaporación de feromona adicional de esos rastros. Por tanto, la regla de actualización de feromona en el SMPH se convierte en:

$$t_{rs} \leftarrow t_{rs} + r \cdot f(C(S_{mejor-global})), \quad \forall a_{rs} \in S_{mejor-global} \tag{15}$$

$$t_{rs} \leftarrow (1-r) \cdot t_{rs}, \quad \forall a_{rs} \in S_{peor-actual} \quad y \quad a_{rs} \notin S_{mejor-global} \tag{16}$$

2. Se realiza una *mutación de los rastros de feromona* para introducir diversidad en el proceso de búsqueda. Para llevarla a cabo, el rastro de feromona asociado a cada una de las transiciones desde cada nodo (por ejemplo, cada fila de la matriz de rastros de feromona) se muta con una probabilidad P_m utilizando cualquier operador de mutación con codificación real.

La propuesta original del SMPH aplicaba un operador que alteraba los rastros de feromona de cada transición mutada añadiendo o restando la misma cantidad en cada iteración. El rango de mutación *mut(it, t_{umbral})*, que depende de la media de los rastros de feromona en las transiciones de la mejor solución global, t_{umbral}, es más suave en las primeras etapas del algoritmo -donde no hay riesgo de estancamiento- y más fuerte en las últimas etapas, donde el peligro de estancamiento es más fuerte:

$$t'_{rs} \leftarrow \begin{cases} t_{rs} + mut(it, t_{umbral}), & si\ a = 0 \\ t_{rs} - mut(it, t_{umbral}), & si\ a = 1 \end{cases} \tag{17}$$

donde a es una valor aleatorio en {0, 1} e *it* es la iteración actual.

3. Como en otros modelos de OCH, el SMPH considera la *reinicialización de los rastros de feromona* cuando se estanca la búsqueda, lo que se lleva a cabo fijando cada rastro de feromona a t_0. En las primeras versiones del algoritmo se comprobaba si el porcentaje de arcos distintos entre la mejor solución y la peor de la población actual era menor que un cierto valor umbral. En versiones más recientes se utiliza un concepto diferente para evaluar si el algoritmo se ha estancado o no: se comprueba el porcentaje de iteraciones del algoritmo sin haber conseguido una mejora en la mejor solución.

El procedimiento `Acciones_del_demonio` es como sigue:

```
1 Procedimiento Acciones_del_demonio
2  para cada Sk hacer búsqueda_local(Sk)
3  Smejor-actual = mejor_solucion(Sk)
4  si (mejor(Smejor-actual,Smejor-global) )
5     Smejor-global = Smejor-actual
6  fin si
```

```
7  para cada arista a_rs ∈ S_mejor-global hacer
8     τ_rs = τ_rs + ρ · f(C(S_mejor-global))
9     suma = suma + τ_rs
10 fin para
11 τ_umbral = sum / |S_mejor-global|
12 S_peor-actual = peor_solución(S_k)
13 para cada arista a_rs ∈ S_peor-global y a_rs ∉ S_mejor-global hacer
14    τ_rs = (1 - ρ) · τ_rs
15 fin para
16 mut = mut (it,τ_umbral)
17 para cada nodo / componente r∈{1, ..., l) hacer
18    z = generar_valor_aleatorio_en_[0,1]
19    si (z ≤ P_m)
20       s = generar_valor_aleatorio_en_[1,…,l]
21       a = generar_valor_aleatorio_en_{0,1}
22       si (a = 0) τ_rs = τ_rs + mut
23       si no τ_rs = τ_rs - mut
24    fin si
25 fin para
26 si (condición_de_estancamiento)
27    para cada arista a_rs hacer τ_rs = τ_0
28 fin si
29 fin Procedimiento
```

Por otro lado, en [21] O. Cordón, I. Fernández de Viana y F. Herrera propusieron el Sistema de Colonias de la Mejor-Peor Hormiga (“*Best-Worst Ant Colony System”*, BWACS), un nuevo modelo de OCH de buen rendimiento donde se incorporan las ideas presentadas anteriormente al SCH aplicadas al problema de la asignación cuadrática (QAP).

8.2 Análisis Estadístico de los Parámetros del SMPH

En [22], se realizó un estudio preliminar sobre las tres componentes características del SMPH a fin de determinar la importancia de cada una de ellas a la hora de obtener soluciones de alta calidad. En esta sección vamos a presentar un estudio estadístico más profundo sobre el comportamiento de las mismas en la resolución del TSP. Para obtener conclusiones fiables, se ha aplicado el test estadístico de Análisis de Varianza (ANOVA) [41, 42], que es una de las herramientas estadísticas más ampliamente utilizadas y que se basa en analizar las medias de las varianzas. Al igual que la regresión, este test se usa para estudiar la relación que existe entre una variable dependiente (variable respuesta) y una o más variables independientes (llamadas factores). Sin embargo, ANOVA se diferencia de la regresión en dos aspectos:

- Las variables independientes son cualitativas.
- No se hace ninguna suposición sobre la naturaleza de la relación existente.

ANOVA nos permite comprobar si la diferencia de la respuesta depende de la variación de uno de los factores o si, por el contrario, ha sido fruto del azar.

Antes de poder aplicar este método, debemos comprobar que las observaciones obtenidas de nuestro experimento cumplen las siguientes condiciones:

- Las observaciones son mutuamente independientes.
- Las observaciones se distribuyen siguiendo una distribución normal.
- Todas tienen la misma varianza.
- Las medias se pueden expresar como combinación lineal.

En toda la experimentación contenida en este artículo, se ha determinado previamente que estas cuatro condiciones se satisfacen.

Con el ANOVA, comprobamos si la hipótesis nula, que presupone que todas las medias son iguales, es cierta o no. El nivel de significancia (*S*) nos indica si la hipótesis se satisface. Si el valor de *S* es menor que 0.05, entonces la importancia del factor es estadísticamente apreciable con un nivel de certeza del 95%.

La experimentación para analizar el comportamiento del SMPH se ha realizado sobre el TSP. A continuación, describimos brevemente dicho problema para pasar después a presentar el estudio de los parámetros utilizados para resolver el problema. Finalmente, se mostrarán los resultados empíricos obtenidos.

8.2.1 El Problema del Viajante de Comercio

El TSP [8] es uno de los problemas de optimización combinatoria más conocidos. Dado un conjunto $N = \{1, ..., n\}$ de ciudades y una matriz $D = [d_{rs}]$, de dimensión $n \times n$, que recoge la distancia existente entre cada par de ciudades (r, s), el TSP se puede formular como la obtención del camino de longitud mínima que pasa por todas las ciudades una sola vez y retorna a la ciudad de origen.

Más formalmente, el TSP se puede representar como un grafo completo con pesos, $G = (N, A)$ donde N es el conjunto de ciudades y A el de arcos. A cada arco se le asigna un valor d_{ij}, que representa la longitud del arco $(i, j) \in A$. El TSP consiste en buscar un circuito Hamiltoniano de coste mínimo en el grafo.

Para este trabajo, usamos las siguiente doce instancias del TSP (todas ellas obtenidas de la *TSPLIB* [89]): *eil51*, *Berlín52*, *brazil58*, *kroa100*, *gr120*, *d198*, *lin318*, *pcb442*, *att532*, *rat783*, *u1060* y *d1291*. Estas doce instancias se pueden dividir en tres familias dependiendo de su tamaño. Cuando hablamos de *instancias pequeñas*, nos referimos a aquellos casos cuyo tamaño no supera las 500 ciudades. Si el tamaño está comprendido entre 500 y 1000, entonces hablaremos de *instancias medianas*. Para tamaños superiores, hablaremos de *instancias grandes*.

8.2.2 Estudio de Parámetros

En esta sección vamos a explicar detalladamente el conjunto de experimentos que hemos realizado para obtener los datos que necesitamos para aplicar el test ANOVA. Con esta experimentación, vamos a tratar de averiguar si los cambios en algunos de los valores de los parámetros de las componentes del SMPH influyen en la bondad de las soluciones alcanzadas. También intentaremos establecer el valor óptimo de dichos parámetros, entendiendo por óptimos aquellos valores con los que el algoritmo consigue las mejores soluciones.

Como ya se ha comentado con anterioridad, el SMPH tiene tres parámetros básicos: la probabilidad de mutación, la fuerza de la mutación y la condición de reinicialización. Para poder estudiar adecuadamente la importancia de cada uno de estos parámetros, es necesario trabajar con distintos valores de los mismos y aplicarlos a distintas instancias del TSP.

Para intentar reducir el número de simulaciones en la medida de lo posible, sin quitar rigor al estudio, hemos optado por:

- Trabajar con un conjunto reducido de instancias representativas del TSP. Concretamente hemos usado una instancia pequeña (*lin318*), otra mediana (*rat783*) y otra grande (*u1060*).
- Limitar el tiempo de cada ejecución. En este primer estudio, no perseguimos obtener los mejores resultados sino estudiar la influencia de los parámetros de las soluciones. Los tiempos de ejecución oscilan entre los 300 sg. para los casos más pequeños y los 900 sg. para los más grandes.
- Para reducir el número de parámetros a utilizar, se ha empleado una mutación distinta a la típica del SMPH. En concreto, se ha optado por aplicar una mutación normal acotada de media cero y de varianza t_{tope}, $mut_n(0, t_{tope})$. Decimos que es acotada porque los valores que genera siguen una distribución normal acotada entre $é\text{-}t_{tope}, t_{tope}ù$. Al igual que el operador de mutación original, la mutación normal se va adaptando al proceso de búsqueda que va cambiando su varianza t_{tope}. En las iteraciones iniciales, t_{tope} tiene un valor muy cercano a t_0 con lo cual la mutación estará limitada en $[-t_0, t_0]$. Conforme el proceso de búsqueda va avanzando, el valor de t_{tope} aumenta, ya que los niveles de feromona de los arcos del grafo visitados por la mejor hormiga global son más elevados que el resto de los arcos. Cuando se produce una reinicialización, el rango de mutación vuelve a ser $[-t_0, t_0]$. En un estudio previo se comprobó que utilizar esta mutación no es estadísticamente peor que emplear el operador de mutación originalmente formulado pero, sin embargo, nos ofrece la ventaja de que no necesitamos definir el parámetro s asociado al operador anterior.

Para el primer factor, el porcentaje de iteraciones sin cambio, hemos analizado , los siguientes niveles: 10%, 20% y 30%. Para P_m, los valores fueron 0.05, 0.1, 0.15, 0.20, 0.25, 0.30 y 0.45. Los valores del resto de parámetros comunes con los que se ha realizado esta primera experimentación están recogidos en la *tabla 1*.

Tabla 1. *Valores de parámetros usados para el TSP*

Parámetro	***Valor***
Número de hormigas	m = 15
Número de ejecuciones	30
Tiempo máximo	300, 600, 900 sg.
Regla de transición	$\alpha = 1$; $\beta = 2$
Cte. evaporación global	$\rho = 0.2$
Búsqueda local	
Número de vecinos	40
Regla de selección	primero mejor
Algoritmo	2-opt

La *tabla 2* contiene los resultados obtenidos por el ANOVA para cada una de las tres instancias del TSP antes comentadas. En esta tabla, la primera columna contiene el factor estudiado, la segunda los grados de libertad (GL), la tercera la suma de los cuadrados (SC), la cuarta la media de los cuadrados (MC), la quinta el valor estadístico F y la última el nivel de significancia (S). Si este último valor es inferior a 0.05, entonces el parámetro afecta con un nivel de confianza del 95%.

Tabla 2. *Resultados del test ANOVA para el TSP*

Lin318					
Factor	**GL**	**SC**	**MC**	**F**	**S**
Mut.	6	142816	23803	2.57	0.018
Rei.	2	17350	8675	0.93	0.393
Rat783					
Factor	**GL**	**SC**	**MC**	**F**	**S**
Mut.	6	48382	8064	20.17	0
Rei.	2	3800	1900	0.93	0.009
U1060					
Factor	**GL**	**SC**	**MC**	**F**	**S**
Mut.	6	39607444	6601241	12.01	0
Rei.	2	4398912	1749456	3.18	0.042

El análisis ANOVA muestra que, en las tres familias de instancias, es la probabilidad de mutación factor que más influye. El nivel de significancia está siempre por debajo de 0.05, con lo queda demostrado que la variación del valor del parámetro de P_m influye en la calidad de la respuesta del algoritmo SMPH.

En lo que respecta al segundo parámetro, su influencia depende de la instancia del TSP que analicemos. En el problema mediano, el valor de *S* está muy por debajo de 0.05, por lo que sí influye. En cambio, para las instancias grandes y pequeñas, *S* es levemente inferior a 0.05 o muy superior a este valor, respectivamente. Así pues, para casos pequeños, los cambios de valor del porcentaje de iteraciones sin mejora no

influyen mientras que para los grandes si hay relación entre su variación y la respuesta del algoritmo.

Para establecer los mejores valores de los parámetros, nos ayudaremos de los datos contenidos en las t*ablas 3 a 5*. En estas tablas, se muestra la media de los resultados obtenidos para cada pareja de parámetros en cada una de las tres instancias del TSP estudiadas.

Tablas 3, 4. *Medias obtenidas para las instancias pequeñas y medianas del TSP*

	Reinicialización		
Mutación	***10%***	***20%***	***30%***
0.05	8884.16	8887.06	8878.2
0.1	8878.23	8869.7	8872.3
0.15	8874.66	8864.63	8870.3
0.20	8873.96	8864.4	8862.03
0.25	8860.03	**8854.7**	8860.53
0.30	8862.63	8856.2	8856.13
0.45	8858.7	8859.1	8856.56

	Reinicialización		
Mutación	***10%***	***20%***	***30%***
0.05	42216.5	42172.9	42161.6
0.1	42155.3	42157.5	42142.9
0.15	42155.6	42184.9	42167.7
0.20	42144.2	42155.4	42143.4
0.25	42134.1	42140.8	42154.8
0.30	42139.7	42151.7	**42127.1**
0.45	42158.8	42143.3	42130

En el caso *lin318*, en el que el único parámetro que influye es la probabilidad de mutación, la mejor combinación es aquella en la que se muta con una probabilidad de 0.25 y la condición de reinicialización es de un 20% de iteraciones consecutivas sin cambio. Obsérvese que en este caso la calidad de la solución va mejorando hasta llegar a $P_m = 0.25$ para después volver a empeorar. Con esta combinación "intermedia"' de valores (ninguno de los factores toma valores extremos), conseguimos mantener un equilibrio adecuado entre la exploración y la explotación que añade cada una de las dos componentes.

Para el *rat783*, la mejor combinación es $P_m = 0.30$ y un 30% como condición de reinicialización, con lo que, en este tipo de problemas, parece más necesario explorar el espacio de búsqueda que explotarlo.

Finalmente, para el *u1060*, la combinación de parámetros óptima es bastante diferente a la de las instancias anteriores. Ahora los mejores valores que toman los parámetros son de 0.15 y 10%. Esta combinación nos sugiere que, para este tipo de problemas, es más acusada la necesidad de realizar una fuerte explotación del espacio de búsqueda.

Tabla 5. *Medias obtenidas para las instancias grandes del TSP*

	Reinicialización		
Mutación	***10%***	***20%***	***30%***
0.05	227106.3	227183.1	227130.5
0.1	227662.8	226869.2	226656.2
0.15	**226321.9**	226712.2	226545.7
0.20	226407.4	226422.6	226333.2
0.25	226668.2	226612.1	226796.7
0.30	226907.1	226907.2	226631.7
0.45	226858.7	226681.7	226565.1

8.2.3 Comparación con otros Algoritmos de OCH

Una vez que hemos obtenido los valores más adecuados para cada uno de los parámetros del SMPH en instancias de distinto tamaño, vamos ha comprobar su buen funcionamiento comparándolo con otros algoritmos de OCH como son el SCH y el SHE. Cada uno de estos algoritmos se ha aplicado a las doce instancias del TSP bajo las mismas condiciones:

- Usando los mismos valores para los parámetros que sean comunes a los algoritmos.
- Empleando una lista de candidatos. Esta es fija, para su construcción sólo se tiene en cuenta la información heurística de la que se dispone al iniciar el proceso de búsqueda.
- Considerando las mismas semillas para los generadores de números aleatorios.
- Empleando el mismo algoritmo de búsqueda local.
- Implementan una versión de SCH y otra de SHE que incluyen la componente de reinicialización del SMPH. A estas dos variantes las denotaremos SCH + Re y SHE + Re, respectivamente.

Aunque existen distintos procedimientos de búsqueda local para el TSP [63], en este artículo se ha optado por usar una versión mejorada del 2-opt [8]. Estas mejoras, encaminadas a incrementar la eficiencia del algoritmo, son:

- Trabajar únicamente con la lista de candidatos.
- Realizar una búsqueda en profundidad y no en anchura.
- Usar "don't look bit" con lo que conseguimos recordar aquellos nodos que ya no son mejorables.

En la *tabla 6* mostramos, de forma resumida, las condiciones en las que se ejecutaron los algoritmos. El tiempo máximo de ejecución oscila entre los 900 y los 3600 sg., dependiendo de si la instancia es pequeña, mediana o grande. Los valores de la mutación y del porcentaje de iteraciones sin cambio variarán dependiendo del tipo de instancia, según lo estudiado en el análisis realizado en la sección anterior.

Tabla 6. Parámetros usados para el TSP

Parámetro	***Valor***
Número de hormigas	m = 15
Número de ejecuciones	10
Tiempo máximo	N_{it} = 900 a 3600
Regla de transición	$\alpha = 1$; $\beta = 2$
Cte. evaporación global	$\rho = 0.2$
SHE	
Hormigas elitistas	m (15)
SCH	
Cte. evaporación local	$q_0 = 0.8$ $\varphi = 0.2$
SMPH	
Prob. de mutación	P_m = 0.3, 0.25, 0.2
Iteraciones sin mejora	0.3, 0.2, 0.3
Búsqueda local	
Número de vecinos	40
Regla de selección	primero mejor
Algoritmo	2-opt

En la *tabla 7* resumimos los resultados obtenidos por cada uno de los algoritmos. Todos los cálculos se han hecho sobre 10 ejecuciones. Dicha tabla está formada por seis columnas cuyos contenidos describimos a continuación:

- en la primera mostramos el nombre del algoritmo aplicado,
- en la segunda aparece el mejor resultado obtenido en las 10 ejecuciones,
- en la tercera, la media de los resultados,
- en la cuarta, la desviación estándar,
- en la quinta está el error medio (proporción asociada a la diferencia entre el mejor costo conocido para el problema y la media de las soluciones alcanzadas por el problema dividida por el costo óptimo) y
- en la última, el número medio de reinicializaciones.

Como se observa en la tabla los mejores resultados se han obtenido aplicando el algoritmo SMPH, que siempre logra un error medio inferior del SHE y el SCH. Además, si comparamos los resultados del SMPH con los publicados, para este mismo algoritmo, en [22], vemos que han mejorado significativamente. Aunque no se haya conseguido mejorar la calidad individual de las soluciones, lo que si hemos logrado es disminuir los errores medios del algoritmo.

Para finalizar, cabe destacar los siguientes aspectos:

- El SMPH es una alternativa interesante a varios de los algoritmos de OCH existentes. Bajo las mismas circunstancias de ejecución, siempre logra mejores soluciones.
- El SMPH es un algoritmo que da buenos resultados incluso cuando los valores de sus parámetros no son los más adecuados, es decir, es un algoritmo bastante robusto.

Adecuar los parámetros del SMPH para resolver un determinado problema mejora la robustez del algoritmo ya que disminuye el error medio de los resultados logrados.

9 Nuevas Tendencias en Optimización Basada en Colonias de Hormigas

Pese a que la OCH es una metaheurística reciente se han desarrollado muchos heurísticas basándose en ella. Aún así es un campo al que le resta bastante tiempo de vida ya que siguen presentándose día a día nuevas tendencias que pretenden mejorar la eficacia de los algoritmos de OCH o mejorar sus tiempos de ejecución. En esta sección presentaremos algunas de las nuevas tendencias más actuales prestando especial atención a la paralelización de las colonias de hormigas.

Tabla 7. *Resultados de la comparativa de los distintos algoritmos*

Problema	**Eil51 (C_{opt} = 426, n = 51)**					**Berlín52 (C_{opt} = 7542), n = 52)**				
Modelo	***Mejor***	***Media***	***Dev.***	***Error***	***#R***	***Mejor***	***Media***	***Dev.***	***Error***	***#R***
SHE	426	426.2	0.44	0.04	-	7542	7542	0	0	-
SCH	426	426.7	0.48	0.48	-	7542	7542	0	0	-
SHE + Re	426	426	426.6	0.26	0	7542	7542	0	0	0
SCH + Re	426	426.5	0.53	0.12	2.4	7542	7542	0	0	0
SMPH	426	426	0	0	0	7542	7542	0	0	0
Problema	**Brazil58 (C_{opt} = 25395, n = 58)**					**Kroal100 (C_{opt} = 21282), n = 100)**				
Modelo	***Mejor***	***Media***	***Dev.***	***Error***	***#R***	***Mejor***	***Media***	***Dev.***	***Error***	***#R***
SHE	25395	25395	0	0	-	21282	21282	0	0	-
SCH	25395	25395	0	0	-	21282	21282	0	0	-
SHE + Re	25395	25395	0	0	0	21282	21282	0	0	0
SCH + Re	25395	25395	0	0	0	21282	21282	0	0	0
SMPH	25395	25395	0	0	0	21282	21282	0	0	0
Problema	**Gr120 (C_{opt} = 6942, n = 120)**					**D198 (C_{opt} = 15780), n = 198)**				
Modelo	***Mejor***	***Media***	***Dev.***	***Error***	***#R***	***Mejor***	***Media***	***Dev.***	***Error***	***#R***
SHE	6942	**6942**	0	0	-	15780	15781	0.70	≅ 0	-
SCH	6942	6946.1	5.49	0.06	-	15780	15784.9	5.67	0.03	-
SHE + Re	6942	**6942**	0	0	0	15780	15781	0.70	≅ 0	0
SCH + Re	6942	6943.8	3.79	0.03	1.1	15780	15782.9	4.31	0.02	2.3
SMPH	6942	**6942**	0	0	0.5	15780	**15780.4**	0.51	≅ 0	1.7
Problema	**Lin318 (C_{opt} = 42029, n = 318)**					**Pcb442 (C_{opt} = 50788), n = 442)**				
Modelo	***Mejor***	***Media***	***Dev.***	***Error***	***#R***	***Mejor***	***Media***	***Dev.***	***Error***	***#R***
SHE	42029	42123.8	67.28	0.22	-	51015	51104.6	82.44	0.63	-
SCH	42029	42230	148.48	0.48	-	50919	51048	75.29	0.53	-
SHE + Re	42029	42105	53.79	0.18	3	50980	51066.8	71.52	0.56	3.2
SCH + Re	42029	42182.4	118.12	0.36	5	50860	51147.5	173.11	0.72	8
SMPH	42029	**42084.1**	98.60	0.13	2	**50788**	**50888.8**	87.83	0.21	6.9
Problema	**Att532 (C_{opt} = 27686, n = 532)**					**Rat783 (C_{opt} = 8806), n = 783)**				
Modelo	***Mejor***	***Media***	***Dev.***	***Error***	***#R***	***Mejor***	***Media***	***Dev.***	***Error***	***#R***
SHE	27745	27823	70.67	0.49	-	8860	8878.6	17.06	0.81	-
SCH	27705	27810.3	64.44	0.45	-	8857	8892.7	20.93	0.97	-
SHE + Re	27793	27825.6	41.59	0.50	3.6	8843	8878.4	32.25	0.81	3.8
SCH + Re	27745	27835	57.56	0.54	7	8875	8899.5	22.33	1.05	7.6
SMPH	**27686**	**27711**	12.16	0.09	7.7	**8816**	**8833.4**	15.37	0.31	9.5
Problema	**U1060 (C_{opt} = 224094, n = 1060)**					**D1291 (C_{opt} = 50801), n = 1291)**				
Modelo	***Mejor***	***Media***	***Dev.***	***Error***	***#R***	***Mejor***	***Media***	***Dev.***	***Error***	***#R***
SHE	231644	232145.8	297.91	3.46	-	51210	51347.2	138.15	1.06	-
SCH	225675	226387.8	668.9	1.01	-	51901	51953.6	60.01	2.21	-
SHE + Re	230360	230806.4	390.8	2.90	2.9	51073	51236.7	119.56	0.85	4.5
SCH + Re	225243	226501	1059.57	1.06	2	51828	51986.8	105.80	2.28	5
SMPH	**225310**	**225721.1**	476.88	0.72	5.3	**50890**	**50986.3**	74.87	0.36	17.2

9.1 Tendencias Actuales en la OCH

La aplicación de algoritmos de OCH a problemas de optimización combinatoria difíciles y el desarrollo de nuevas y mejores variantes algorítmicas de la OCH son las dos tendencias más activas en cuanto a investigación se refiere en el área. En concreto, la aplicación de la OCH a una creciente variedad de problemas (en muchos casos NP-duros) es la principal vía de contribuciones en este campo. Merecen especial atención los trabajos sobre aplicaciones en problemas de optimización multi-objetivo y problemas de optimización dinámicos. Entre las primeras aproximaciones a los *problemas de optimización multi-objetivo* se incluye el enfoque basado en dos colonias propuesto por Gambardella, Taillard y Agazzi para el enrutamiento de vehículos con ventanas de tiempo, donde se consideran dos objetivos ordenados de manera jerárquica [45]. Mariano y Morales han considerado la aplicación de la OCH a problemas de optimización multi-objetivo en el diseño de redes de irrigación. En su aproximación se asigna una colonia a cada objetivo y éstos se ordenan según su importancia [70]. Iredi, Markle y Middendorf han explorado la aplicación de la OCH para encontrar soluciones Pareto-optimales para un problema de secuenciación bi-criterio en una sola máquina [61].

Por otro lado uno de los mayores éxitos de los algoritmos de OCH se ha obtenido en aplicaciones a problemas altamente dinámicos como los de enrutamiento en redes de comunicaciones (el AntNet de Di Caro y Dorigo es la variante con un mejor rendimiento [29]). Para dichas aplicaciones los algoritmos de OCH son muy apropiados porque hacen corresponder de manera adecuada sus características con las de los problemas como, por ejemplo, las transiciones estocásticas entre estados en la que sólo está disponible información local. Otra dirección en la que se encamina la investigación en problemas dinámicos es la aplicación de la OCH en variantes de problemas de optimización NP-duros clásicos cuyas características cambian con el tiempo, como el TSP o el QAP. Como ejemplos pueden presentarse los trabajos de Guntsch y Middendorf [52, 51, 53] y de Eyckelhof y Snoek [38].

Con respecto a las técnicas algorítmicas en general, una tendencia importante actual es el desarrollo de técnicas híbridas que combinan ideas de algoritmos exactos y metaheurísticas. En la OCH, Maniezzo ha desarrollado un algoritmo particular con esta hibridación: la búsqueda en árbol aproximada no determinística en árboles (*"Aproximate Nondeterministic Tree-Search"*, ANTS) [67] que combina los algoritmos de OCH con otras técnicas de umbrales inferiores de la programación matemática. En concreto, se calculan los umbrales inferiores en el coste de completar una solución parcial son calculadas y se utilizan como información heurística sobre lo idóneo que es continuar añadiendo componentes. Esto ofrece ventajas como la posible eliminación de extensiones de soluciones parciales si conllevan un coste mayor que la mejor solución encontrada hasta el momento.

Al ser la OCH cada vez más y más usada, estudiada y desarrollada [25], muchos investigadores han pasado a prestar atención a las razones de su éxito para así llegar a una mejor comprensión de su comportamiento en la búsqueda. Stützle y Hoos han enlazado estas causas del éxito de la OCH con la composición del espacio de

búsqueda de los problemas de optimización combinatoria. En concreto, la OCH parece comportarse bastante bien en problemas que muestran una gran correlación entre la calidad de las soluciones y la distancia hasta el óptimo global (medida por la llamada correlación valor de adaptación-distancia [62]) [98]. Estudios del comportamiento de la OCH en problemas sencillos como la búsqueda del camino más corto o problemas de permutaciones (resolubles en tiempo polinomial) muestran la importancia de algunas de sus características de diseño, como la actualización de la feromona basada en la calidad de las soluciones o el uso de diferentes estrategias para construir soluciones [34, 72]. La dinámica de los algoritmos OCH ha sido estudiada por Merkle y Middendorf [74]. Por último, Blum, Samples y Zlochin han demostrado que definiciones particulares de los rastros de feromona pueden llevar a una situación inesperada en la que el rendimiento del SH puede incluso degradarse en tiempo de ejecución [11]. Cuando se usan mejores algoritmos de OCH que el SH, Blum y Sampels también muestran que la manera en la que se definen los rastros de feromona tiene (como cabía esperarse) una gran influencia en el comportamiento de los algoritmos de OCH [10].

9.2 Paralelización de Colonias de Hormigas

Actualmente se están dedicando esfuerzos a investigar procedimientos que incrementen la velocidad de proceso de los algoritmos OCH así como mejoren la calidad de las soluciones obtenidas utilizando procesamiento en paralelo. Es preciso distinguir dos grandes grupos dentro de los algoritmos paralelos basados en colonias de hormigas: los de paralelización real y los de paralelización simulada. Otra gran distinción en este tipo de algoritmos son los que ofrecen una paralelización a nivel de Hormigas o una paralelización a nivel de Colonias de Hormigas. Como veremos a continuación, la paralelización a nivel de hormigas se lleva a cabo utilizando técnicas de paralelización real, mientras que la paralelización a nivel de colonias se lleva a cabo mediante paralelización simulada.

En los algoritmos de paralelización real, se utilizan distintos nodos de cómputo (usualmente ordenadores conectados en una red) para intentar solucionar un problema concreto. Este tipo de algoritmos tratan esencialmente de mejorar los tiempos de cómputo o bien incrementar la potencia de cálculo disponible. Los algoritmos de paralelización simulada realizan una paralelización dentro del mismo nodo de cómputo, lo que no permite a priori disminuir los tiempos de computo. Sin embargo ofrecen unas características especiales que intentan esencialmente mejorar la calidad de las soluciones obtenidas.

Dentro del primer grupo de algoritmos existen numerosas variantes. Debido a que de manera natural la metaheurística OCH presenta una estructura paralela, donde los agentes de cómputo (las hormigas artificiales) se mueven a priori de manera independiente y asíncrona por el espacio de búsqueda, se puede asignar a cada unidad de cálculo el trabajo correspondiente de la generación de las soluciones por parte de las hormigas. Para ello, se pueden utilizar distintas topologías, aunque la más empleada se corresponde con una estructura de tipo *maestro / esclavo* entre las

unidades de cómputo [16, 77, 78, 93, 103], donde un procesador central lleva el control de la matriz de feromona (y realiza la actualización de la información memorística) y los procesadores esclavos se encargan de la construcción de las soluciones (sobre las que pueden aplicar opcionalmente una búsqueda local). Este tipo de algoritmos en paralelo han evolucionado hasta estructuras algo más complejas (estructuras de árbol) donde se emplean ciertos nodos de cómputo únicamente para realizar búsquedas locales de las soluciones encontradas -como ya se comentó anteriormente, la búsqueda local se ha convertido en un elemento casi indispensable para la obtención de buenas soluciones en la OCH, y cada vez se emplea más tiempo de cálculo para realizar estás tareas de optimización local- (*figura 2*). Recientemente también se ha presentado la adaptación de un algoritmo de OCH para ejecutarse en arrays de procesadores reconfigurables [73].

Como se puede comprobar, este tipo de paralelización no ofrece, en principio, grandes cambios en la estructura de los algoritmos, simplemente distribuye las tareas de cómputo de manera más o menos eficiente entre diversos procesadores, lo que nos permite obtener soluciones en un tiempo menor (o realizar un mayor número de iteraciones del algoritmo en la misma cantidad de tiempo).

En este tipo de paralelizaciones hay que prestar mucha atención a los tiempos de comunicación entre procesadores. Es bien conocido que cualquier proceso que se desee paralelizar no incrementa su velocidad de manera directamente proporcional al número de nodos que se empleen, sino que el límite se encuentra en una cota bastante inferior, regida por la llamada *ley de Amdahl* [58]. Este límite viene impuesto normalmente por los tiempos de comunicación y sincronización que existen en los algoritmos paralelos. Por lo tanto, los algoritmos que minimicen la cantidad de datos a transmitir y el número de sincronizaciones necesarias mejorarán de una manera más eficiente conforme se incremente el número de procesadores involucrados en la tarea. De hecho, de las dos estructuras comentadas con anterioridad, la de árbol presenta un mejor comportamiento debido a que la cantidad de datos a transmitir es menor: para las búsquedas locales no hace falta usualmente la información memorística sino sólo la heurística, que suele estar a disposición de los procesadores al comienzo del algoritmo, mientras que para la construcción de

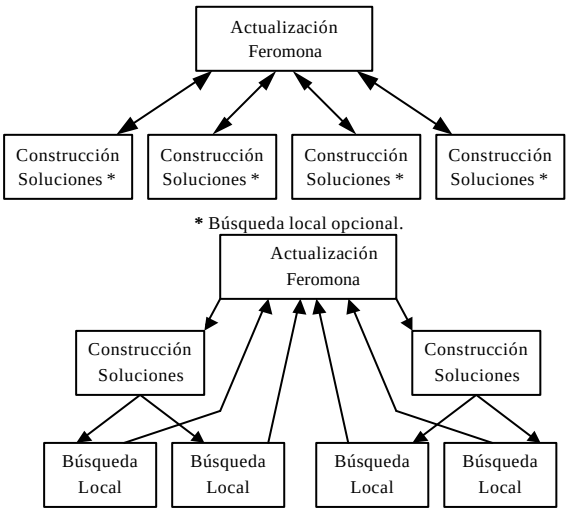

Fig 2. *Distintas topologías típicas entre procesadores para paralelizar algoritmos basados en colonias de hormigas*

soluciones el procesador maestro debe encargarse de repartir la información memorística entre sus nodos esclavos.

Precisamente, ese límite impuesto por las comunicaciones es el motivo fundamental por el que no se suelen realizar paralelizaciones reales de algoritmos de OCH a nivel de Colonias de Hormigas, ya que la cantidad de información que hay que retransmitir entre los nodos puede ser muy elevada, lo que restaría eficiencia al proceso creando un cuello de botella difícil de evitar.

Hay que analizar en detalle cada posible algoritmo a fin de determinar si la cantidad de información que se transmite hará posible su implementación usando distintos procesadores (los algoritmos en los que sólo se intercambia información sobre la mejor hormiga si que son susceptibles a una paralelización entre varias unidades de cómputo). Sin embargo, casi todas las experimentaciones que se han realizado en la práctica han sido llevadas a cabo en un único procesador (paralelización simulada) a fin de comparar las bondades de las distintas variantes.

Los algoritmos de paralelización de Colonias de Hormigas construyen normalmente varios espacios de búsqueda diferentes, es decir, varias colonias de hormigas distintas (aunque la información heurística suele coincidir en todas, la información memorística evolucionará de manera más o menos independiente en cada colonia). Cada cierto tiempo, las distintas colonias comparten algo de información con sus vecinas, esperando de esta manera colaborar mutuamente en mejorar las soluciones generadas en las distintas colonias.

El tipo de información que se comparte, entre qué colonias se comparte y de qué manera se interpreta esa información permite crear numerosas variantes de algoritmos de OCH de colonias paralelas. Entre los tipos de información que se comparten, se puede incluir información sobre hormigas particulares (usualmente la mejor hormiga de cada colonia) o sobre la matriz de feromona de las colonias, por ejemplo.

Las colonias de hormigas tienen que intercambiar la información siguiendo alguna estructura concreta. Normalmente, se emplean distintas variantes, todas ellas tomadas del campo de los Algoritmos Genéticos paralelos [17]: modelo de isla, estructura en anillo o emparejamiento aleatorio, entre otros.

Respecto al uso que hacen las distintas colonias de la información compartida, éste varía dependiendo principalmente del tipo de información que se comparte: cuando se intercambia la información sobre la mejor hormiga, ésta se emplea usualmente a modo de hormiga elitista (refuerzo de los arcos por los que viajó dicha hormiga). Cuando se intercambian matrices de feromona completas, existen diversas posibilidades. Una de las más sencillas es utilizar la mejor de todas las matrices de feromona en todas las colonias, aunque se han propuesto técnicas más complejas que realizan sumas de dos matrices de feromona o incluso medias de feromonas basadas en cruces heurísticos [59, 105]. Este tipo de medias pretenden ampliar el espacio de búsqueda reforzando los caminos buenos que aparecen en varias matrices y restándole

importancia a los caminos que solo están marcados como buenos en una de ellas (*Figura 3*).

Fig 3. *Ejemplo de media de matrices de feromona. Los arcos más oscuros son los que contienen mayor cantidad de feromona. Como se puede comprobar tras hacer la media de las matrices, los arcos del circuito más corto tienen más feromona con lo que sería más probable conseguir la mejor solución*

Un problema importante al que se enfrentan los algoritmos que intercambian la matriz de feromona completa es que no existe ningún método sencillo para evaluar la bondad de las mismas. Una solución casi inmediata es asignar una bondad a la matriz que sea proporcional a la mejor solución obtenida con la misma. Sin embargo, como se deduce fácilmente, no es una medida de gran fiabilidad puesto que casualmente una matriz de baja calidad puede producir una solución bastante buena (no hay que olvidar que la metaheurística OCH ofrece un sistema de generación de soluciones que es en esencia aleatorio). Para un estudio más en profundidad sobre paralelización simulada de colonias de hormigas, así como de las distintas alternativas que existen y del análisis de algunos resultados empíricos obtenidos, recomendamos la lectura de [4].

Hay que hacer notar por último que este tipo de paralelizaciones sí que representan un cambio sustancial en los algoritmos, con el objetivo de mejorar la calidad de las soluciones obtenidas, frente a los anteriores que únicamente trataban de disminuir los tiempos de cómputo utilizando más capacidad de cálculo.

10 Comentarios Finales

Hoy en día, la OCH es una metaheurística bien definida y con un buen rendimiento que se aplica para resolver cada vez un mayor número de problemas de optimización combinatoria complejos. En este trabajo, hemos revisado las ideas subyacentes a esta aproximación, que nos llevan desde la inspiración biológica hasta la metaheurística OCH. Se han descrito muchas de las aproximaciones actuales y se han resumido algunos aspectos teóricos así como resultados sobre diferentes tópicos como las relaciones con otras metaheurísticas. Además, hemos identificado algunas tendencias actuales en el campo, intentando esclarecer el posible futuro en el desarrollo de la

OCH. Por otro lado se ha presentado un análisis del comportamiento del SMPH más en profundidad sobre el problema TSP, demostrando que las partes fundamentales del algoritmo se complementan para obtener soluciones de alta calidad.

Referencias

[1] E. H. L. Aarts, J. H. M. Korst, y P. J. M. van Laarhoven. Simulated annealing. En E. H. L. Aarts y J.K. Lenstra, editores, *Local Search in Combinatorial Optimization*, páginas 91-120. John Wiley & Sons, Chichester, 1997.

[2] E. H. L. Aarts y J. K. Lenstra, editores. *Local Search in Combinatorial Optimization*. John Wiley & Sons, Chichester, 1997.

[3] R. Alcalá, J. Casillas, O. Cordón, y F. Herrera. Improvement to the cooperative rules methodology by using the Ant Colony System algorithm. *Mathware & Soft Computing*, 8: 3, páginas 321-335, 2001.

[4] S. Alonso, O. Cordón, I. Fernández de Viana, F. Herrera. Análisis de distintas vertientes para la paralelización de los algoritmos de Optimización basada en Colonias de Hormigas. *Actas del Segundo Congreso Español de Metaheurísticas, Algoritmos Evolutivos y Bioinspirados (MAEB03)*, páginas 160-167, Gijón, 2003.

[5] T. Bäck. *Evolutionary Algorithms in Theory and Practice*. Oxford University Press, New York, NY, 1996.

[6] T. Bäck, D. Fogel, Z. Michalewicz, editores, *Handbook of Evolutionary Computation*, 1998

[7] S. Baluja and R. Caruana. Removing the genetics from the standard genetic algorithm. En A. Prieditis y S. Russel, editores, *Proceedings of the Twelfth International Conference on Machine Learning (ML-95)*, pages 38-46. Morgan Kaufmann Publishers, Palo Alto, CA, 1995.

[8] J. L. Bentley, Fast algorithms for geometric travelling salesman problem, *ORSA Journal on Computing*, 4: 4, páginas 387-411, 1992.

[9] M. Birattari, T. Stützle, L. Paquete, y K. Varrentrapp. A racing algorithm for configuring metaheuristics. En W.B. Langdon y otros, editor, *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, páginas 11-18. Morgan Kaufmann Publishers, San Francisco, CA, 2002.

[10] C. Blum y M. Sampels. When model bias is stronger than selection pressure. En *Proceedings of PPSN-VII, Seventh International Conference on Parallel Problem Solving from Nature*, volume 2439 of *Lecture Notes in Computer Science*, páginas 893-902. Springer Verlag, Berlín, Alemania, 2002.

[11] C. Blum, M. Sampels, y M. Zlochin. On a particularity in model-based search. En W.B. Langdon y otros, editores, *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, páginas 35-42. Morgan Kaufmann Publishers, San Francisco, CA, 2002.

[12] E. Bonabeau, M. Dorigo, y G. Theraulaz. *Swarm Intelligence: From Natural to Artificial Systems*. Oxford University Press, New York, NY, 1999.

[13] G. Brassard y P. Bratley. *Fundamentals of Algorithmics*. Prentice Hall, Englewood Cliffs, NJ, 1996.

[14] B. Bullnheimer, R. F. Hartl, y C. Strauss. An improved ant system algorithm for the vehicle routing problem. *Annals of Operations Research*, 89, páginas 319-328, 1999.

[15] B. Bullnheimer, R. F. Hartl, y C. Strauss. A new rank-based version of the Ant System: A computational study. *Central European Journal for Operations Research and Economics*, 7: 1, páginas 25-38, 1999.

[16] B. Bullnheimer, G. Kotsis, y C. Strauss. Parallelization strategies for the Ant System. En R. De Leone, A. Murli, P. Pardalos, y G. Torradlo, editores, *High Performance Algorithms and Software in Nonlinear Optimization*, volumen 24 de *Applied Optimization*, páginas 87-100. Kluwer Academic Publishers, Dordrecht, Holanda, 1998.

[17] E. Cantú-Paz. A survey of parallel genetic algorithms. *Calculateurs Parallèles Réseaux of Systèmes Répartis*, 10: 2, páginas 141-171, 1998.

[18] J. Casillas, O. Cordón, y F. Herrera. Learning fuzzy rules using ant colony optimization algorithms. En M. Dorigo, M. Middendorf and T. Stützle, editores, *Abstract proceedings of ANTS2000 - From Ant Colonies to Artificial Ants: A Series of International Workshops on Ant Algorithms*, páginas 13-21. IRIDIA, Université Libre de Bruxelles, Belgium, 2000.

[19] J. Casillas, O. Cordón, y F. Herrera. Different approaches to induce cooperation in fuzzy linguistic models under the COR methodology. En B. Bouchon-Meunier, J. Gutiérrez-Rios, L. Madalena, y R.R. Yager, editores, *Technologies for Constructing Intelligent Systems 1.* Tasks, páginas 321-334. Physica-Verlag, 2002.

[20] A. Colorni, M. Dorigo, V. Maniezzo, y M. Trubian. Ant System for job-shop scheduling. *JORBEL - Belgian Journal of Operations Research, Statistics and Computer Science*, 34: 1, páginas 39-53, 1994.

[21] O. Cordón, I. Fernández de Viana, y F. Herrera. Analysis of the Best-Worst Ant System and its variants on the QAP. En M. Dorigo, G. Di Caro, y M.Sampels, editores, *Proceedings of ANTS2002 - From Ant Colonies to Artificial Ants: Third International Workshop on Ant Algorithms*, volume 2463 of *Lecture Notes in Computer Science*, páginas 228-234. Springer Verlag, Berlín, Alemania, 2002.

[22] O. Cordón, I. Fernández de Viana, y F. Herrera. *Analysis of the Best Worst Ant System and its variants on the TSP*. Mathware & Soft Computing, 9: 2-3, páginas 177-192, 2002.

[23] O. Cordón, I. Fernández de Viana, F. Herrera, y L. Moreno. A new ACO model integrating evolutionary computation concepts: The Best-Worst Ant System. En M. Dorigo, M. Middendorf, y T. Stützle, editores, *Abstract proceedings of ANTS2000 - From Ant Colonies to Artificial Ants: A series of International Workshops on Ant Algorithms*, páginas 22-29. IRIDIA, Université Libre de Bruxelles, Belgium, 2000.

[24] O. Cordón, F. Herrera, L. Moreno. Integración de Conceptos de Computación Evolutiva en un Nuevo Modelo de Colonias de Hormigas. *VIII Conferencia de la Asociación Española*

para la Inteligencia Artificial, (Seminario Especializado en Computacion Evolutiva), Murcia (España), 1999, Vol. II, páginas 98-105.

[25] O. Cordón, F. Herrera, T. Stützle, A Review on Ant Colony Optimization Metaheuristic: Basis, Models and New Trends. Mathware & Soft Computing, 9: 2-3, páginas 141-175, 2002.

[26] D. Costa and A. Hertz. Ant can colour graphs. *Journal of the Operational Research Society*, 48, páginas 295-305, 1997.

[27] L. M. de Campos, J. M. Fernández-Luna, J. A. Gámez, y J. M. Puerta. Ant colony optimization for learning bayesian networks. *International Journal of Approximate Reasoning*, 31: 3, páginas 291-311, 2002.

[28] L. M. de Campos, J. A. Gámez, y J. M. Puerta. Learning bayesian networks by ant colony optimisation: searching in two different spaces. *Mathware & Soft Computing*, 9: 2-3, páginas 251-268, 2002.

[29] G. Di Caro and M. Dorigo. AntNet: Distributed stigmergetic control for communications networks. *Journal of Artificial Intelligence Research*, 9:317-365, 1998.

[30] M. Dorigo y G. Di Caro. The Ant Colony Optimization meta-heuristic. En D. Corne, M. Dorigo, y F. Glover, editores, *New Ideas in Optimization*, páginas 11-32. McGraw Hill, London, UK, 1999.

[31] M. Dorigo, G. Di Caro, y L. M. Gambardella. Ant algorithms for discrete optimization. *Artificial Life*, 5: 2, páginas 137-172, 1999.

[32] M. Dorigo y L. M. Gambardella. Ant Colony System: A cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1: 1, páginas 53-66, 1997.

[33] M. Dorigo, V. Maniezzo, y A. Colorni. The Ant System: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics - Part B*, 26: 1, páginas 29-41, 1996.

[34] M. Dorigo y T. Stützle. An experimental study of the simple ant colony optimization algorithm. En N. Mastorakis, editor, *2001 WSES International Conference on Evolutionary Computation (EC'01)*, páginas 253-258. WSES Press, 2001.

[35] M. Dorigo y T. Stützle. *Ant Colony Optimization*. MIT Press, Cambridge, MA. En prensa.

[36] M. Dorigo y T. Stützle. The ant colony optimization metaheuristic: Algorithms, applications and advances. En F. Glover and G. Kochenberger, editores, *Handbook of Metaheuristics*, páginas 251-285. Kluwer Academic Publishers, 2003.

[37] M. Dorigo, M. Zlochin, N. Meuleau, y M. Birattari. Updating ACO pheromones using stochastic gradient ascent and cross-entropy methods. En S. Cagnoni, J. Gottlieb, E. Hart, M. Middendorf, y G. R. Raidl, editores, *Applications of Evolutionary Computing, Proceedings of Evo Workshops 2002*, volume 2279 of *Lecture Notes in Computer Science*, páginas 21-30. Springer Verlag, Berlín, Alemania, 2002.

[38] C.J. Eyckelhof y M.Snoek. Ant systems for a dynamic TSP: Ants caught in a traffic jam. En M. Dorigo, G. Di Caro, y M.Sampels, editores, *Proceedings of ANTS2002 - From Ant Colonies to Artificial Ants: Third International Workshop on Ant Algorithms*, volume 2463 of *Lecture Notes in Computer Science*, páginas 88-99. Springer Verlag, Berlín, Alemania, 2002.

[39] T. A. Feo y M. G. C. Resende. Greedy randomized adaptative search procedures. Journal of Global Optimization, 6, páginas 109-133, 1995.

[40] P. Festa y M. G. C. Resende. GRASP: An annotated bibliography. En P. Hansen y C. C. Ribeiro, editores, *Essays and Surveys on Metaheuristics*. Kluwer Academic Publishers, 2001.

[41] R. A. Fisher, Theory of Statistical Estimation. *Proceedings of the Cambridge Philoshophical Society*, 22, páginas 700-725, 1925.

[42] R.A. Fisher, The Comparison of Samples with Possibly Unequal Variances, *Annals of Eugenics*, 9, páginas 174-180, 1936.

[43] L. M. Gambardella and M. Dorigo. Ant-Q: A reinforcement learning approach to the traveling salesman problem. En A. Prieditis y S. Russell, editores, *Proceedings of the Twelfth International Conference on Machine Learning (ML-95)*, páginas 252-260. Morgan Kaufmann Publishers, Palo Alto, CA, 1995.

[44] L. M. Gambardella y M. Dorigo. Ant Colony System hybridized with a new local search for the sequential ordering problem. *INFORMS Journal on Computing*, 12: 3, páginas 237-255, 2000.

[45] L. M. Gambardella, È. D. Taillard, y G. Agazzi. MACS-VRPTW: A multiple ant colony system for vehicle routing problems with time windows. En D. Corne, M. Dorigo y F. Glover, editores, *New Ideas in Optimization*, páginas 63-76. McGraw Hill, London, UK, 1999.

[46] L. M. Gambardella, È.D. Taillard, y M. Dorigo. Ant colonies for the quadratic assigment problem. Journal of the Operational Research Society, 50: 2, páginas 167-176, 1999.

[47] M. R. Garey y D. S. Jonson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, San Francisco, CA, 1979.

[48] F. Glover and G. Kochenberger, editores. Handbook of Metaheuristics. Kluwer Academic Publishers, 2003.

[49] F. Glover y M. Laguna. Tabu Search. Kluwer Academic Publishers, Boston, MA, 1997.

[50] S. Goss, S. Aron, J. L. Deneubourg, y J. M. Pasteels. Self-organized shortcuts in the Argentine ant. *Naturwissenschaften*, 76, páginas 579-581, 1989.

[51] M. Guntsch y M. Middendorf. An ant colony optimization approach to dynamic TSP. En L. Spector y otros editor, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, páginas 860-867. Morgan Kaufmann Publishers, San Francisco, CA, 2001.

[52] M. Guntsch and M. Middendorf. Pheromone modification strategies for ant algorithms applied to dynamic TSP. En E.J.W. Boers y otros editor, *Applications of Evolutionary Computing: Proceedings of Evo Workshops 2001*, volume 2037 of *Lectures in Computer Science*, páginas 213 - 222. Springer Verlag, Berlín, Alemania, 2001.

[53] M. Guntsch y M. Middendorf. Applying population based ACO to dynamic optimization problems. En M. Dorigo, G. Di Caro, y M.Sampels, editores, *Proceedings of ANTS2002 - From Ant Colonies to Artificial Ants: Third International Workshop on Ant Algorithms*, volume 2463 of *Lecture Notes in Computer Science*, páginas 111-122. Springer Verlag, Berlín, Alemania, 2002.

[54] W. J. Gutjahr. A Graph-based Ant System and its convergence. *Future Generation Computer Systems*, 16: 8, páginas 873-888, 2000.

[55] W. J. Gutjahr. ACO algorithms with guaranteed convergence to the optimal solution. *Information Processing Letters*, 82: 3, páginas 145-153, 2002.

[56] B. Hajek. Cooling schedules for optimal annealing. *Mathematics of OR*, 13, páginas 311-329, 1988

[57] P. Hansen y N. Mladenovic. An introduction to variable neighborhood search. En S. Voss, S.Martello, I. H. Osman, y C. Roucairol, editores, *MetaHeuristics - Advances and Trends in Local Search Paradigms for Optimization*, páginas 433-458. Kluver Academic Publishers, Dordrecht, Holanda, 1999.

[58] J. L. Hennesy, D. A. Patterson, Computer Architecture, A Quantitative Approach, páginas 40-42, Morgan Kaufmann Publishers, 2003.

[59] F. Herrera, M. Lozano, J. L. Verdegay. Dynamic and Heuristic Fuzzy Connectives-Based Crossover Operators for Controlling the Diversity and Convergence of Real Coded Genetic Algorithms. *Int. Journal of Intelligent Systems*, 11, páginas 1013-1041, 1996.

[60] J. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, 1975.

[61] S. Iredi, D. Merkle, y M. Middendorf. Bi-criterion optimization with multi colony ant algorithms. En R. Zitzler y otros, editor, *Evolutionary Multi-Criterion Optimization, First International Conference (EMO'01)*, volume 1993 of *Lecture Notes in Computer Science*, páginas 359-372. Springer Verlag, Berlín, Alemania, 2001.

[62] T. Jones y S. Forrest. Fitness distance correlation as a measure of problem difficulty for genetic algorithms. En L.J. Eshelman, editor, *Proceedings of the 6th International Conference on Genetic Algorithms*, páginas 184-192. Morgan Kaufman, 1995.

[63] D. S. Johnson, L. A. McGeoch. The Traveling Salesman Problem: A Case Study in Local Optimization, E.H.L. Arts, J.K. Lenstra, editores, *Local Search in Combinatorial Optimization*, páginas 215-310, John Wiley & Sons, 1997.

[64] S. Kirkpatrick, C. D. Gelatt Jr., y M. P. Vecchi. Optimization by simulated annealing. *Science*, 220, páginas 671-680, 1983.

[65] P. Larrañaga y J. A. Lozano, editores. *Estimation of distribution Algorithms: A New Tool for Evolutionary Computation*. Kluwer Academic Publishers, Dordrecht, Holanda, 2001.

[66] H. R. Lourenço, O. C. Martin, y T. Stützle. Iterated local search. En F. Glover and G. Kochenberger, editores, *Handbook of Metaheuristics*, páginas 321-353. Kluwer Academic Publishers, 2003.

[67] V. Maniezzo. Exact and approximate nondeterministic tree-search procedures for the quadratic assigment problem. *INFORMS Journal on Computing*, 11: 4, páginas 358-369, 1999.

[68] V. Maniezzo y A. Colorni. The Ant System applied to the quadratic assignment problem. *IEEE Transactions on Data and Knowledge Engineering*, 11: 5, páginas 769-778, 1999.

[69] V. Maniezzo, A. Colorni, y M. Dorigo. The Ant System Applied to the quadratic assigment problem. Technical Report IRIDIA/94-28, IRIDIA, Université Libre de Bruxelles, Belgium, 1994.

[70] C. E. Mariano y E. Morales. MOAQ: An Ant-Q algorithm for múltiple objective optimization problems. En W. Banzhaf y otros, editor, *Genetic and Evolutionary Computation Conference (GECCO-99)*, volumen 1, páginas 894-901. Morgan Kaufmann Publishers, San Francisco, CA, 1999.

[71] B. Mendoza y C.A. Coello. An approach based on the use of the Ant System to design combinatorial logic circuits. *Mathware & Soft Computing*, 9: 2-3, páginas 235-250, 2002.

[72] D. Merkle y M. Middendorf. On the behaviour of ant algorithms: Studies on simple problems. En *Proceedings of the 4^h Metaheuristics International Conference (MIC'2001)*, páginas 573-577, Oporto, Portugal, 2001.

[73] D. Merkle y M. Middendorf. Fast ant colony optimization on runtime reconfigurable processor arrays. *Genetic Programming and Evolvable Machines*, 3: 4, páginas 345-361, 2002.

[74] D. Merkle y M. Middendorf. Studies on the dynamics of ant colony optimization algorithms. En W. B. Langdon y otros, editores, *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, páginas 105-112. Morgan Kaufmann Publishers, San Francisco, CA, 2002.

[75] N. Meuleau y M. Dorigo. Ant colony optimization and stochastic gradient descent. *Artificial Life*, 8: 2, páginas 103-121, 2002.

[76] Z. Michalewicz y D. B. Fogel. *How to Solve It: Modern Heuristics*. Springer Verlag, Berlín, Alemania, 2000.

[77] R. Michel y M. Middendorf. An Island model based Ant system with lookahead for the shortest supersequence problem. En A. E. Eiben, T. Bäck, M. Schoenauer, y H.-P. Schwefel, editores, *Proceedings of PPSN-V, Fifth International Conference on Parallel Problem Solving from Nature*, volume 1498 of *Lecture notes in Computer Science*, páginas 692-701. Springer Verlag, Berlín, Alemania, 1998.

[78] M. Middendorf, F. Reischle, y H. Schmeck. Multi colony ant algorithms. *Journal of Heuristics*, 8: 3, páginas 305-320, 2002.

[79] T. Mitchell. *Machine Learning*. McGraw Hill, Boston, MA, 1997.

[80] N. Monmarché, E. Ramat, G. Dromel, M. Slimane, y G. Venturini. On the similarities between AS, BSC and PBIL: Toward the birth of a new metaheuristic. Informe Técnico E3i, 215, Ecole d'Ingénieurs en Informatique pour l'Industrie, Université de Tours, 1999.

[81] P. Moscato. Memetic algorithms: A short introduction. En D. Corne, M. Dorigo, y F. Glover, editores, *New Ideas in Optimization*, páginas 219-234. McGraw Hill, London, Reino Unido, 1999.

[82] H. Mühlenbein y D. Schlierkamp-Voosen. The science of breeding and its application to the breeder genetic algorithm (BGA). *Evolutionary Computation*, 1: 4, páginas 335-360, 1993.

[83] I. H. Osman and J. P. Kelly, editores. Meta-Heuristics: Theory & Applications. Kluwer Academic Publishers, Boston, MA, 1996.

[84] C. H. Papadimitriou y K. Steiglitz. *Combinatorial Optimization - Algorithms and Complexity*. Prentice Hall, Englewood Cliffs, NJ, 1982.

[85] R. S. Parpinelli, H.S. Lopes, y A. A. Freitas. An ant colony based system for data mining: application to medical data. En *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, páginas 791-798. Morgan Kaufmann Publishers, San Francisco, CA, 2001.

[86] R. S. Parpinelli, H. S. Lopes, y A.A. Freitas. Data mining with an Ant Colony Optimization algorithm. *IEEE Transaction on Evolutionary Computation*, 2002, 6 4, páginas 321-332, 2002.

[87] J. M. Pasteels, J.-L. Deneubourg, y S. Goss. Self-organization mechanisms in ant societes (I): Trail recruitment to newly discovered food sources. *Experientia Supplementum*, 54, páginas 155-175, 1987.

[88] C. Reeves, editor. *Modern Heuristic Techniques for Combinatorial Problems*. McGraw Hill, London, UK, 1995.

[89] G. Reinelt, TSPLIB, página web accesible en http://www.iwr.uni-heidelberg.de/iwr/comopt/soft/TSPLIB95/TSPLIB.html.

[90] H. Robbins y H. Monroe. A stochastic approximation meted. *Annals of Mathematics and Statistics*, 22, páginas 400-407, 1951.

[91] R. Schoonderwoerd, O. Holland, J. Bruten, y L. Rothkrantz. Ant-based load balancing in telecommunications networks. *Adaptative Behavior*, 5: 2, páginas 169-207, 1996.

[92] T. Stützle. An ant approach to the flow shop problem. En *Proceedings of the 6th European Congress on Intelligent Techniques & Soft Computing (EUFIT'98)*, volumen 3, páginas 1560-1564. Verlag Mainz, Wissenschaftsverlag, Aachen, Alemania, 1998.

[93] T. Stützle. Parallelization strategies for ant colony optimization. En A. E. Eiben, T. Bäck, M. Schoenauer, y H.-P. Schwefel, editores, *Proceedings of PPSN-V, Fifth International Cenference on Parallel Problem Solving from Nature*, volume 1498 of *Lecture Notes in Computer Science*, páginas 722-731. Springer Verlag, Berlín, Alemania, 1998.

[94] T. Stützle. *Local Search Algorithms for Combinatorial Problems: Analysis, Improvements, and New Applications*, volume 220 of *DISKI*. Infix, Sankt Augustin, Alemania, 1999.

[95] T. Stützle y M. Dorigo. ACO algorithms for the traveling salesman problem. En K. Miettinen, M. M. Mäkelä, P. Neittaanmäki, y J. Périaux, editores, *Evolutionary Algorithms in Engineering and Computer Science*, páginas 163-183. John Wiley & Sons, Chichester, Reino Unido, 1999.

[96] T. Stützle y M. Dorigo. A short convergence proof for a class of Ant Colony Optimization algorithms. *IEEE Transactions on Evolutionary Computation*, 6: 4, páginas 358-365, 2002.

[97] T. Stützle y H. H. Hoos. The MAX-MIN Ant System and local search for the traveling salesman problem. En T. Bäck, Z. Michalewicz, y X. Yao, editores, *Proceedings of the 1997 IEEE International Conference on Evolutionary Computation (ICEC'97)*, páginas 309-314. IEEE Press, Piscataway, NJ, 1997.

[98] T. Stützle y H. H. Hoos. MAX-MIN Ant System. *Future Generation Computer Systems*, 16: 8, páginas 889-914, 2000.

[99] T. Stützle y H. H. Hoos. Improving the Ant system: A detailed report on the MAX-MIN Ant System. Technical Report AIDA-96-12, FG Intellektik, FB Informatik, TU Darmstadt, Alemania, Agosto 1996.

[100] T. Stützle y S. Linke. Experiments with variants of ant algorithms. *Mathware & Soft Computing*, 9: 2-3, páginas 193-207, 2002.

[101] G. Syswerda. Simulated crossover in genetic algorithms. En *Foundations of Genetic Algorithms, Second Workshop (FOGA 2)*, páginas 309-314, 1993.

[102] É. D. Taillard y L. M. Gambardella. Adaptative memories for the quadratic assigment problem. Informe Técnico IDSIA-87-97, IDSIA, Lugano, Switzerland, 1997.

[103] E.-G. Talbi, Olivier Roux, C. Fonlupt, D. Robillard, Parallel Ant Colonies for the quadratic assigment problem. *Future Generation Computer Systems*, 17, páginas 441-449, 2001.

[104] S. Voss, S. Martello, I.H. Osman, y C. Roucairol, editores. *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*. Kluwer Academic Publishers, Boston, MA, 1999.

[105] A. Wright. Genetic Algorithms for Real Parameter Optimization. *Foundations of Genetic Algorithms, First Workshop on the Foundations of Genetic Algorithms and Clasified Systems*, páginas 205-218, G.J.E. Rowling (Ed.), Morgan Kaufmann, Los Altos, CA, 1990.

[106] M. Zlochin, M. Birattari, N. Meuleau, y M. Dorigo. Model-based search for combinatorial optimization. Informe Técnico TR/IRIDIA/2001-16, IRIDIA, Université Libre de Bruxelles, Bélgica, 2001.