Análisis de distintas vertientes para la paralelización de los algoritmos de Optimización basada en Colonias de Hormigas[1]

Sergio Alonso[2], Oscar Cordón[2], Iñaki Fernández de Viana[2], Francisco Herrera[2]

Resumen - Este trabajo tiene como objetivo hacer una comparativa de los principales modelos de paralelización de algoritmos de Optimización Basados en Colonias de Hormigas (OCH), así como presentar una nueva variante en la que se introducirán elementos tomados de otro tipo de algoritmos bioinspirados como son los Algoritmos Genéticos. Asimismo, se pretende analizar cuáles deben ser los aspectos de los algoritmos de OCH paralelos que deben potenciarse con el fin de obtener los mejores resultados.

Palabras clave - Optimización Basada en Colonias de Hormigas, Sistema de Hormigas, Paralelización, Viajante de Comercio.

I. INTRODUCCIÓN

En el transcurso de los últimos años las *Heurísticas basadas en la naturaleza* o *Algoritmos bioinspirados* [1], [2] ha sido uno de los campos de investigación en el desarrollo de metaheurísticas [17] que más expectativas ha levantado. Algunas de las metaheurísticas más importantes que se han desarrollado son los *Algoritmos Genéticos*, *Enfriamiento Simulado* y los *Algoritmos de Optimización Basados en Colonias de Hormigas* (OCH), entre otras.

La OCH corresponde a una de las metaheurísticas más recientes [3], [4]. Los algoritmos desarrollados dentro de esta línea de investigación tratan de simular el modo en el que las hormigas encuentran y transportan la comida siguiendo un camino de longitud mínima entre la fuente de alimento y el hormiguero.

Existen numerosas propuestas de algoritmos de OCH, entre las que caben destacar el *Sistema de Hormigas* (SH) [4], el *Sistema de Colonias de Hormigas* (SCH) [5], el *Sistema de Hormigas Max-Min* [8] y el *Sistema de la Mejor-Peor Hormiga* (SMPH) [6], [7].

Recientemente la metaheurística OCH ha comenzado a verse enriquecida por distintas variantes donde se introducen nuevos elementos a los algoritmos con la finalidad de mejorar las soluciones obtenidas o bien de disminuir los tiempos de cálculo necesarios para la ejecución de los mismos. Uno de estos elementos que más variantes de algoritmos está produciendo es la *paralelización*.

Nuestro objetivo es analizar las principales tendencias en la paralelización de los algoritmos de OCH, así como los aspectos de esta paralelización que permiten obtener mejoras en el rendimiento de los mismos. En resumen, pretendemos responder a la siguiente pregunta: ¿es la paralelización de los algoritmos de OCH una buena vía para mejorar los resultados que se obtienen?. Y en caso afirmativo, ¿cómo debe llevarse a cabo esa paralelización?.

El trabajo comienza presentando los fundamentos de la metaheurística OCH junto con una de las primeras propuestas que se hicieron para ella, el Sistema de Hormigas (*SH*), y una de sus variantes más extendidas, el *SH Elitista*. Posteriormente se presentarán las dos vertientes principales de paralelización hasta el momento. A continuación nos centraremos en aquella que está diseñada para mejorar las soluciones obtenidas. Una vez presentadas todas las alternativas clásicamente usadas en la bibliografía, se propondrá una nueva variante de paralelización que se inspira en ciertos aspectos de los Algoritmos Genéticos. Por último pasaremos al análisis de los resultados empíricos obtenidos, así como a las conclusiones alcanzadas.

II. ALGORITMOS OCH

A. Introducción a la metaheurística OCH

La metaheurística OCH [3] puede englobarse dentro de las metaheurísticas bioinspiradas. Esta metaheurística se compone de diversos algoritmos compuestos por poblaciones de agentes cooperativos que imitan a las hormigas reales. En concreto imitan el comportamiento de estos insectos a la hora de encontrar y transportar comida. Durante el proceso de búsqueda de comida las hormigas van depositando en el suelo una sustancia denominada feromona. Posteriormente una hormiga es capaz de oler dicha hormona y dirigir su búsqueda -y por tanto la búsqueda de la colonia en conjunto- en función de las cantidades depositadas. La manera en la que una hormiga determina la dirección que debe

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

[2] Departamento de Ciencias de la Computación e Inteligencia Artificial, E.T.S. Ingeniería Informática, C/ Periodista Daniel Saucedo Aranda s/n 18071 Granada (España), zerjio79@hotmail.com, ocordon@decsai.ugr.es, ijfviana@teleline.es, herrera@decsai.ugr.es

seguir al encontrar una intersección en su camino es sencilla: elige de manera probabilística pero ponderada por la cantidad de feromona depositada en cada ramificación. El movimiento continuado de todas las hormigas provoca que los mejores caminos (los más cortos) hasta la fuente de alimento posean una mayor cantidad de feromona al ser frecuentados por un mayor número de hormigas. Por el contrario los caminos más largos son progresivamente abandonados, por lo que la feromona depositada en ellos termina por evaporarse. De esta manera se termina por encontrar un camino de longitud mínima entre el hormiguero y la comida.

Para poder utilizar este tipo de algoritmos es necesario que las soluciones del problema que deseamos resolver puedan ser representadas como un camino en un grafo.

Cada uno de los arcos de nuestro grafo *(r,s)* almacenará dos tipos de información distinta:

- *Información heurística* (τ_{rs}): Depende exclusivamente del problema que deseamos resolver. Normalmente es calculada antes de comenzar la ejecución de nuestro algoritmo, y representa la calidad a priori de dicho arco. En el caso de las hormigas naturales esta información se corresponde con la dificultad o distancia que posee el tramo que tiene que recorrer una hormiga (depende únicamente de la topografía del terreno).
- *Información memorística* (η_{rs}): Esta información se va modificando con la ejecución de nuestro algoritmo y depende del número de hormigas que han circulado por dicho arco y de la bondad de las soluciones que obtuvieron dichas hormigas. Para las hormigas naturales esta información se corresponde con las deposiciones de feromona que existen en el terreno. A partir de ahora a esta información se le llamará *feromona del arco (r,s)*.

El funcionamiento básico de un algoritmo de OCH es el siguiente:

1. Se inicializa la información heurística del grafo.
2. Se posiciona una población de *m* hormigas en el grafo.
3. Cada una de las hormigas construye progresiva e independientemente del resto distintos recorridos dentro del grafo siguiendo una regla de transición de estados que depende de la información existente (normalmente basada tanto en la información heurística como en la información memorística).
4. Se evalúa la bondad de las soluciones obtenidas y opcionalmente se aplica una técnica de búsqueda local para mejorarlas.
5. Se actualiza la información memorística.
6. Si no se verifica la condición de finalización volvemos al punto 2.

Existen elementos que son propios del problema que se va a resolver como:

- La inicialización de la información heurística.
- La inicialización de las hormigas.
- La obtención de los nodos alcanzables por una hormiga en cada etapa de su recorrido.

y otros que dependen de la implementación concreta del algoritmo de OCH escogido:

- Qué regla de transición de estados se utiliza.
- Cómo se actualiza la información memorística.

B. El modelo SH

En la literatura existen diversas propuestas para la regla de transición de estados y la actualización de la información memorística. Uno de los modelos característicos de esta metaheurística es el *Sistema de Hormigas*, desarrollado por Dorigo, Maniezzo y Colorni [4].

De hecho, el Sistema de Hormigas fue la primera propuesta de heurística OCH que se realizó. Actualmente existen distintas variantes del SH (una de las cuales se discutirá más tarde), pero todas ellas tienen en común las siguientes características:

1. Todos los arcos del mapa *(r,s)* tienen asociada una información de encaminamiento:

$$b_{rs} = [\tau_{rs}]^{\alpha} \cdot [\eta_{rs}]^{\beta} \quad \forall s \in N(r)$$

donde *N(r)* es el conjunto de vecinos de *r*, α y β son constantes que ponderan la importancia que queremos darle a la información heurística y memorística respectivamente.

2. A cada uno de los posibles movimientos que una hormiga k puede realizar en un instante se le asocia una probabilidad de selección:

$$p_{rs}^{k} = \frac{b_{rs}}{\sum_{u \in N(r)} b_{ru}} \quad \forall s \in N(r)$$

Una vez calculadas las probabilidades, la regla de transición selecciona una de las opciones aleatoriamente.

3. Cuando cada hormiga ha generado una solución al problema, S_k, todas contribuyen a la actualización de información memorística. El nuevo valor de feromona de cada arco *(r,s)* será:

$$\tau_{rs} \leftarrow (1-\rho)\cdot\tau_{rs} + \sum_{k=1}^{m}\Delta\tau_{rs}^{k}$$

donde

$$\Delta\tau_{rs}^{k} = \begin{cases} f(C(S_k)), & si(r,s)\in S_k \\ 0 & otro\ caso \end{cases}$$

y $f(C(S_k))$ es la cantidad de feromona depositada por la hormiga k. Dicha cantidad depende de la calidad de la solución generada por la hormiga, $C(S_k)$. ρ es una constante de evaporación global. A esta forma de actualizar la feromona una vez calculada la solución completa al problema se le denomina *actualización en línea a posteriori* o *global*.

C. Una variante sobre el SH: SH elitista

Una de las variantes del SH más comúnmente utilizadas es aquella que incluye las denominadas hormigas *elitistas* [4]. Estas hormigas elitistas refuerzan, mediante una deposición adicional de feromona en cada iteración del algoritmo, los arcos que forman parte de la mejor solución encontrada hasta el momento.

Esta actualización adicional provoca una convergencia más rápida del algoritmo, permitiendo alcanzar mejores soluciones más rápidamente.

Si nuestro sistema de hormigas posee e hormigas elitistas y la mejor solución hasta el momento es S_{MS}, la actualización de feromona en cada arco será:

$$\tau_{rs} \leftarrow (1-\rho)\cdot\tau_{rs} + \sum_{k=1}^{m}\Delta\tau_{rs}^{k} + e\cdot\Delta\tau_{rs}^{MS}$$

donde

$$\Delta\tau_{rs}^{MS} = \begin{cases} f(C(S_{MS})), & si(r,s)\in S_{MS} \\ 0 & otro\ caso \end{cases}$$

y $f(C(S_{MS}))$ es la cantidad de feromona depositada por las hormigas elitistas, que depende de la calidad de la mejor solución encontrada hasta el momento, $C(S_{MS})$.

III. PARALELIZACIÓN DE LOS ALGORITMOS OCH

Los algoritmos de OCH ofrecen de manera relativamente sencilla diversas formas de paralelización que nos permiten obtener mejores resultados o tiempos de ejecución menores. Existen básicamente dos vertientes de paralelización a niveles completamente distintos: *paralelización de hormigas* (paralelización de grano fino) y *paralelización de colonias de hormigas* (paralelización de grano grueso).

A. Paralelización de hormigas

Los algoritmos de OCH presentan de por sí una posibilidad relativamente sencilla para poder explotar el paralelismo: las hormigas de la colonia construyen sus soluciones de manera independiente, teniendo en cuenta únicamente la información del grafo. Por lo tanto, es posible distribuir los procedimientos de construcción de soluciones entre varios procesadores distintos con el fin de reducir los tiempos de ejecución del algoritmo.

En la literatura se encuentran diversas aproximaciones a este tipo de paralelismo, donde típicamente se presenta una estructura *maestro / esclavo* entre procesadores [9], [10]. El procesador maestro se encarga de actualizar la información memorística en cada iteración del bucle así como de suministrar la información que necesiten los procesadores esclavos. Éstos a su vez se dedican exclusivamente a la construcción de las nuevas soluciones (*Fig. 1*).

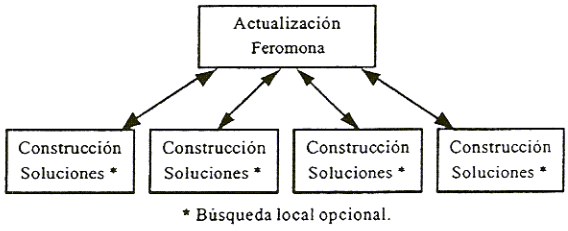


Fig 1: Paralelización siguiendo un esquema *Maestro / Esclavo*

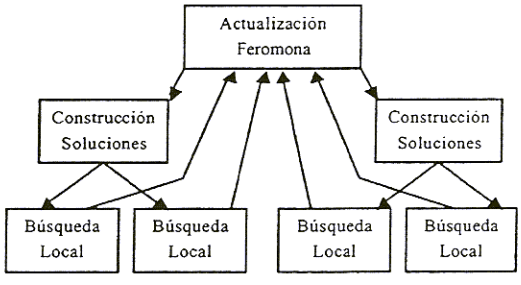


Fig 2: Paralelización siguiendo un esquema de *árbol*

Sin embargo, este tipo de paralelización no introduce ningún cambio sustancial en la estructura de los algoritmos (las soluciones se construyen en procesadores distintos del que realiza la actualización, pero todos los pasos del algoritmo se realizan en el mismo orden), por lo que tampoco se consigue una mejora de la solución, sino un decremento en el tiempo de cálculo del algoritmo. Sin embargo es importante reseñar que, al aumentar la velocidad de ejecución del algoritmo, siempre es muy probable que para un mismo tiempo de ejecución esta paralelización obtenga mejores resultados (salvo los casos en los que el algoritmo se haya estancado).

Por supuesto, siempre debemos tener en cuenta la ley de Amdahl según la cual no podremos en ningún caso pensar que con este tipo de estructuras conseguiremos mejoras de tiempo proporcionales al número de procesadores involucrados [18] -pese a que las hormigas trabajen en paralelo de manera independiente siempre hay sincronizaciones necesarias y parte del código que no puede paralelizarse, como la actualización de feromona-.

B. Paralelización de colonias de hormigas

La segunda vía de paralelizar los algoritmos de OCH es utilizar varias colonias de hormigas distintas y en principio independientes. Esta aproximación no está diseñada para conseguir una mejora en el tiempo de ejecución de los algoritmos, pese a que puede ser implementada con relativa facilidad en múltiples procesadores, sino para obtener soluciones de mayor calidad. Esta técnica ha sido utilizada previamente en la literatura para resolver algunos problemas específicos.

Existen numerosas formas de llevar a cabo esta paralelización en función, básicamente, de cómo comparten la información las distintas colonias entre sí, y lo que quizás sea más importante, *la información que comparten*. A continuación estudiamos varios de ellos.

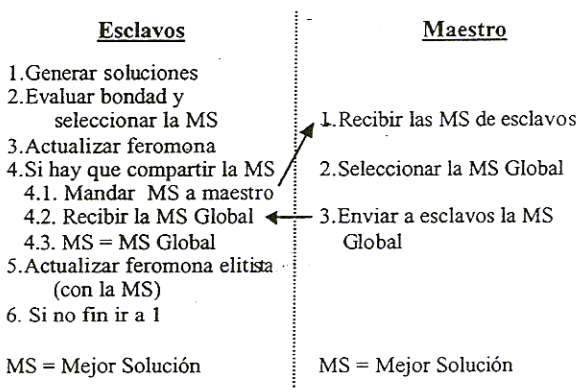
Ejecuciones independientes

Este es el algoritmo de paralelización de colonias más sencillo. Existen t colonias distintas que intentan solucionar el mismo problema sin compartir ninguna información entre ellas. Es equivalente a ejecutar t veces el algoritmo básico no paralelizado. Por su sencillez de implementación (incluso en varias máquinas distintas de maneras simultanea) es uno de los más usados en la bibliografía [11], [12].

Compartición de la mejor hormiga encontrada

Debido a que muchos de los algoritmos de OCH utilizan la mejor solución encontrada hasta el momento para realizar la actualización de feromona, (por ejemplo el *SH elitista*), dicha solución es una clara candidata a ser compartida por las colonias que se ejecutan en paralelo. Normalmente las t colonias se ejecutan independientemente pero cada cierto tiempo (cada *it* iteraciones del algoritmo, por ejemplo) comparten la mejor solución obtenida hasta el momento. Existen a su vez diversas maneras para compartir dicha solución: usando una configuración *maestro / esclavo* o una *estructura en anillo* [12].

En la configuración *maestro / esclavo*, todas las colonias ponen en común las mejores soluciones encontradas hasta el momento, entre las cuales se selecciona la mejor y se envía de vuelta a cada colonia para sustituir su antigua mejor solución. A continuación se describe más detalladamente el algoritmo para el caso del *SH elitista*:

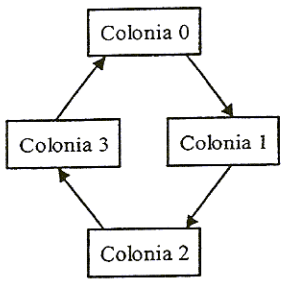


A la hora de efectuar la compartición se ha de establecer un emparejamiento entre procesadores par ala comunicación. Existen distintas variantes, tomadas del campo de los Algoritmos Genéticos paralelos [15].

Cuando se usa una *estructura en anillo* las colonias se distribuyen de tal manera que todas posean un vecino siguiente y uno anterior. En el momento de intercambiar la mejor solución, cada colonia ofrece su mejor solución a su vecino siguiente y recibe la mejor solución del vecino anterior. Cada colonia selecciona la solución de mayor calidad entre la suya propia y la recibida desde el vecino anterior (*Fig. 3*).

Compartición de la matriz de feromona

Aunque clásicamente la única información que se ha compartido entre colonias que se ejecutan en paralelo es la información sobre la mejor solución encontrada hasta el momento, existe la posibilidad de compartir la matriz de feromona completa.



Colonia i

1.Generar soluciones
2.Evaluar bondad y seleccionar la MS
3.Actualizar feromona
4.Si hay que compartir la MS
  4.1. Mandar la MS a (i + 1)
  4.2. Recibir la MSA de (i – 1)
  4.3. MS = mejor (MS, MSA)
5.Actualizar feromona elitista (con la MS)
6. Si no fin ir a 1

MS = Mejor Solución
MSA = Mejor Solución de la colonia Anterior

Fig 3: Esquema de compartición de la mejor solución en anillo

En la bibliografía se encuentran pocas referencias con respecto a este tipo de traspaso de información y, en general, las alusiones que se hacen a esta técnica suelen ser desalentadoras [13]. Sin embargo, hemos creído conveniente hacer un estudio más completo al respecto para comprobar si realmente es poco ventajoso que las colonias compartan la información sobre la feromona depositada en los arcos del grafo.

Existen, al igual que en el caso de compartir la mejor solución, distintas aproximaciones para compartir la información memorística de las colonias. Dos de ellas son aplicar las mismas configuraciones de *maestro / esclavo* y en *anillo* anteriormente descritas.

El único aspecto que hay que aclarar para que este método tenga sentido es cómo se evalúa la calidad de las matrices de feromona. Una posible medida de la bondad de las matrices puede ser la calidad de la mejor solución de la colonia. Sin embargo no parece muy adecuado considerar que una matriz de feromona sea mejor que otra porque simplemente haya conseguido una hormiga mejor (recordemos que el proceso de generación de soluciones tiene una gran componente aleatoria, con lo que matrices de baja calidad pueden obtener en ocasiones buenas soluciones). Por otra parte, es muy sencillo que diversas colonias tengan la misma mejor solución, debido principalmente a la información que comparten entre sí. Para paliar este inconveniente, proponemos como medida de bondad de las matrices de feromona la ordenación lexicográfica de los parámetros *calidad de la mejor solución* y *calidad media de las soluciones de la última generación calculada: [C(MS), C(SM)]*.

Esto es, para comparar dos matrices de feromona m_1 y m_2 utilizaríamos la siguiente función:

```
función compara(m1, m2) {
  si C(MS(m1)) > C(MS(m2))   devuelve m1;
  si C(MS(m1)) < C(MS(m2))   devuelve m2;

  si C(SM(m1)) > C(SM(m2))
    devuelve m1
  si no
    devuelve m2;
}
```

donde `C(MS(x))` representa la calidad de la mejor solución generada a partir de la matriz de feromona `x` y `C(SM(x))` es la calidad media de las soluciones de la última generación calculada usando la matriz `x`.

IV. Modelo de Medias de Matrices de Feromona

Una de las primeras propuestas en paralelización de algoritmos de OCH con compartición de matrices de feromona sugería la suma de las matrices de feromona como un posible camino para encontrar mejores soluciones. Sin embargo, las experimentaciones llevadas a cabo, empleando únicamente con dos colonias distintas, no presentaban resultados verdaderamente satisfactorios.

En esta sección vamos a presentar un nuevo algoritmo paralelo que hace uso de las medias de matrices para mejorar la calidad de las soluciones encontradas por nuestras colonias en paralelo. Para el desarrollo de este modelo se han utilizado ideas presentes en los Algoritmos Genéticos [14], [15].

El algoritmo paralelo funciona en principio como los anteriormente propuestos. Las t colonias de hormigas se ejecutan en paralelo y cada it iteraciones se produce el intercambio de información entre ellas. Por el contrario el intercambio que realizamos es algo más complejo. Se emparejan de manera aleatoria las distintas colonias y se realiza una media ponderada entre sus matrices de feromona. Dicha matriz media se convierte en la nueva matriz de feromona de ambas colonias.

La media ($\overline{m}$) de las matrices de feromona m_1 y m_2 se lleva a cabo usando un cruce heurístico [19], [20] que viene determinado por la siguiente expresión:

$$\tau_{rs}^{\overline{m}} \leftarrow \mu \cdot \tau_{rs}^{m_1} + (1-\mu) \cdot \tau_{rs}^{m_2} \quad \forall\, arco\,(r,s)$$

donde

$$\mu = \frac{C(MS(m_1))}{C(MS(m_1)) + C(MS(m_2))}$$

La idea de emparejar aleatoriamente las colonias persigue como objetivo crear una mayor diversidad en las soluciones que se construyan (puesto que los rastros de feromona que existen en cada población serán distintos). Adicionalmente evitamos que el algoritmo se estanque rápidamente en una solución, ya que no todas las colonias poseen información sobre dicha solución. Asimismo, esta idea puede potenciar los tramos buenos contenidos en varias matrices de feromona distintas: si un arco de nuestro grafo contiene mucha cantidad de feromona en dos matrices que se van a cruzar (porque ese tramo es muy usado por las hormigas de ambas colonias), en la matriz resultado seguirá teniendo una gran cantidad de feromona, con lo que las futuras hormigas que circulen por el mapa tendrán una posibilidad alta de escoger dicho arco.

Así pues, podemos imaginar este modelo como un Algoritmo Genético sencillo, donde la población (con pocos individuos) son las matrices de feromona, cuyo operador de cruce es la media ponderada, y donde no existe mutación.

Una diferencia fundamental con los Algoritmos Genéticos es que estos actúan cruzando y mutando soluciones del problema con el fin de obtener nuevas (y mejores) soluciones. En cambio nosotros estamos utilizando un cruce sobre matrices de feromona, que son uno de los elementos necesarios para generar soluciones (no hay que perder de vista que las soluciones son generadas por las hormigas y que las matrices de feromona no son las soluciones propiamente dichas del problema).

V. Experimentación y análisis de resultados

En la *Tabla I* se exponen todos los algoritmos que se han utilizado para realizar el estudio comparativo. En total son nueve los algoritmos implementados, aunque los cuatro primeros son similares entre sí. Lo único que varía es el número de colonias que se ejecutan en paralelo. Con estos cuatro pretendemos hacer una primera evaluación sobre la importancia que tiene la paralelización. Los demás algoritmos intentarán darnos una idea sobre cuáles de las ideas expuestas con anterioridad proporcionan soluciones mejores.

A. Experimentación

Para evaluar cada uno de los algoritmos hemos escogido el problema del *Viajante de Comercio* (TSP). Este problema consiste en minimizar la longitud de un circuito hamiltoniano que pase por todos los nodos de un grafo únicamente una vez. En concreto se han utilizado las instancias *eil101* y *gr120*, disponibles en la biblioteca TSPLIB [16].

Algoritmo	Descripción
SHE_I_T1	SH Elitista Indep. (sin comunic.) 1 colonia
SHE_I_T2	SH Elitista Indep. (sin comunic.) 2 colonias
SHE_I_T4	SH Elitista Indep. (sin comunic.) 4 colonias
SHE_I_T8	SH Elitista Indep. (sin comunic.) 8 colonias
SHE_H_M/S	SH Elitista Inter. Hormigas Maestro / Esclavo
SHE_H_Ring	SH Elitista Inter. Hormigas estructura Anillo
SHE_M_M/S	SH Elitista Inter. Matrices Maestro / Esclavo
SHE_M_Ring	SH Elitista Inter. Matrices estructura Anillo
SHE_M_Ave	SH Elitista Cruce de Matrices por Medias

Tabla I: Algoritmos usados para el estudio comparativo.

Todos los algoritmos han sido ejecutados bajo las mismas condiciones para evitar incluir arbitrariedad en los posteriores análisis. Concretamente:

- Se ha usado la misma parametrización, solo para los parámetros comunes a los distintos problemas
- Se ha empleado una lista de candidatos fija, donde únicamente se tiene en cuenta la información heurística de cada problema.
- Se han utilizado las mismas semillas para los generadores de datos aleatorios.

En ningún caso se han utilizado técnicas de búsqueda local para poder asegurar que la bondad de los distintos algoritmos proviene de la paralelización de los mismos y no se debe a un proceso en principio ajeno a las heurísticas utilizadas.

En todos los casos el tiempo de ejecución ha sido el mismo. Igualmente todas las ejecuciones se han llevado a cabo con el mismo procesador. De hecho, en todos los casos la paralelización ha sido simulada en un único procesador. En la *Tabla II* se muestran las condiciones usadas para todos los algoritmos.

Parámetro	Valor
Número de Hormigas global	$m = 40$
Número de Colonias	$t = 8$ (salvo en los SHE_I)
Hormigas por Colonia	m / t
Hormigas Elitistas	$e = m / t$
Tiempo de Ejecución	*900 segundos*
Parámetros regla transición	$\alpha = 1, \beta = 2$
Cte. Evaporación global	$\rho = 0.2$
Búsqueda Local	*NO*
Número de ejecuciones	*15*
Intercambio de Información	cada $it = 10$
Tamaño Lista de Candidatos	20

Tabla II: Parámetros usados para el TSP.

B. Análisis de los resultados

En las *Tablas III* y *IV* se resumen los resultados obtenidos por cada uno de los algoritmos

presentados. Todos los cálculos se han hecho en base a 15 ejecuciones distintas. Las tablas está formada por 5 columnas cuyos contenidos describimos a continuación:

- Nombre del algoritmo.
- Error medio: proporción asociada a la diferencia entre el mejor costo conocido para el problema y la media de soluciones obtenidas por el algoritmo en cuestión dividida por el costo óptimo:

$$E_m = \frac{Media\ Soluciones - Óptimo}{Óptimo} \times 100$$

- Mejor resultado obtenido en las 15 ejecuciones.
- Media de los resultados.
- Desviación estándar.

Problema	eil101 (C_{opt} = 629, n = 101)			
Algoritmo	E_m	*Mejor*	*Media*	*Desv.*
SHE_I_T1	3.19	640	649.07	5.65
SHE_I_T2	2.61	639	645,47	4.39
SHE_I_T4	2.38	637	644	4.31
SHE_I_T8	2.38	631	644	5.52
SHE_H_M/S	2.56	629	645.13	6.64
SHE_H_Ring	2.21	630	642.93	7.59
SHE_M_M/S	2.71	634	646.07	7.38
SHE_M_Ring	2.71	633	646.07	6.93
SHE_M_Ave	1.68	631	639.6	6.72

Tabla III: Resultados para el problema *eil101*.

Problema	gr120 (C_{opt} = 6942, n = 120)			
Algoritmo	E_m	*Mejor*	*Media*	*Desv.*
SHE_I_T1	5.64	7181	7333.5	104.4
SHE_I_T2	5.53	7216	7326.2	76.2
SHE_I_T4	6.03	7175	7360.8	173.6
SHE_I_T8	5.51	7200	7324.2	63
SHE_H_M/S	5.61	7164	7331.3	89
SHE_H_Ring	4.53	7119	7256.1	78.1
SHE_M_M/S	4.91	7190	7282.7	70.4
SHE_M_Ring	4.29	7121	7239.9	81.1
SHE_M_Ave	3.31	7013	7172.4	69.3

Tabla IV: Resultados para el problema *gr120*.

Una de las primeras conclusiones que podemos sacar de los datos conseguidos tras la ejecución de los algoritmos es que la paralelización de Colonias de Hormigas parece una buena directriz para la mejora de las soluciones obtenidas por los algoritmos de OCH.

Si comparamos los resultados de los cuatro primeros algoritmos, conforme el número de colonias aumenta, los resultados que se van obteniendo son mejores en líneas generales. Esta mejora de las soluciones se debe atribuir a que las distintas colonias amplían el espacio de búsqueda evitando estancamientos prematuros del algoritmo. Una vez obtenida esa primera conclusión, tratamos de ver las posibles mejoras que ofrece la comunicación entre las distintas colonias.

Los resultados nos muestran que la estructura *maestro / esclavo* no ha ofrecido mejoras sustanciales y en algunos casos incluso ha empeorado los resultados, tanto para el intercambio de mejores soluciones como para el de matrices de feromona.

Sin embargo, la *estructura en anillo* si que mejora en términos generales las soluciones. La explicación a este fenómeno es que una estructura en anillo evita de nuevo el estancamiento de los algoritmos en soluciones concretas, pues permite que las matrices de feromona de las distintas colonias evolucionen de manera más independiente.

Respecto a la utilidad de intercambiar únicamente la mejor solución encontrada o la matriz de feromona completa, podemos afirmar que en términos generales no se puede determinar que alguno de los dos métodos sea más eficaz (pese a que en el primer problema el intercambio de matrices no haya obtenido resultados buenos, en el segundo supera por un amplio margen al intercambio de hormigas). Aun así, no hay que olvidar que el intercambio de matrices de feromona, especialmente cuando los algoritmos sean ejecutados en una arquitectura de procesadores distribuidos, es una operación más costosa en tiempo, debido principalmente al tamaño de las mismas.

Finalmente nos queda reseñar que el último algoritmo ha sido el que mejores resultados ha obtenido, probablemente porque la operación de cruce de matrices de feromona permite conservar los rastros de feromona más importantes a la vez que se añaden caminos alternativos de calidad procedentes de otras colonias de hormigas. Un ejemplo simplificado de este proceso es el que se ilustra en la *Figura 4*.

Retomemos ahora las preguntas que nos hacíamos al principio del trabajo:

- ¿Es la paralelización de los algoritmos OCH una buena vía para mejorar los resultados que se obtienen? Sí, los resultados obtenidos en las distintas aproximaciones en paralelo siempre han superado las aproximaciones secuenciales.

- ¿Cómo debe llevarse a cabo esa paralelización? De todas las alternativas paralelas, la que mejores resultados ha obtenido ha sido la que implementa un cruce de matrices de feromona con el fin de diversificar las soluciones obtenidas. Lo que si es importante reseñar es que parece que el intercambio de información entre las colonias que trabajan en paralelo es un elemento clave para conseguir una mejora sustancial en nuestros algoritmos.

VI. CONCLUSIONES

En este trabajo se ha realizado un estudio sobre las distintas posibilidades de paralelización que presentan los algoritmos OCH. Después de presentar las dos vertientes de paralelización básicas (paralelización de Hormigas y paralelización de Colonias de Hormigas), se ha pasado a un estudio más profundo de esta segunda aproximación al problema, por ser la que está diseñada con vistas a mejorar las soluciones obtenidas.

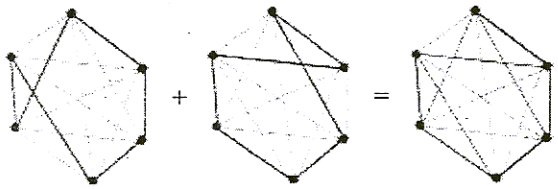


Fig. 4: Ejemplo de media de matrices de feromona. Los arcos más oscuros son los que contienen mayor cantidad de feromona. Como se puede comprobar tras hacer la media de las matrices, los arcos del circuito más corto tienen más feromona con lo que sería más probable conseguir la mejor solución.

Tras una presentación de las distintas modalidades de paralelización de Colonias de Hormigas, se ha realizado un pequeño estudio empírico usando para ello el problema TSP, en el que ha quedado reflejado que este tipo de paralelización es una buena vía para mejorar las soluciones obtenidas. En concreto, la nueva propuesta que hemos realizado, que utiliza ideas extraídas de otras metaheurísticas bioinspiradas como son los Algoritmos Genéticos, ha mostrado un comportamiento bastante positivo.

En próximos trabajos queda pendiente analizar más instancias del TSP (especialmente casos más grandes) e intentar resolver otros problemas, así como investigar nuevas posibilidades de inclusión de otros operadores típicamente usados en Algoritmos Genéticos para modificar las matrices de feromona de las distintas colonias (como son, por ejemplo, los operadores de mutación).

REFERENCIAS

[1] E. Bonabeon, M. Dorigo, G. Theraulaz. Swarm Intelligence. From Natural to Artificial Systems. Oxford University Press, 1999.

[2] Peter J. Bentley. Digital Biology. Headline 2001.

[3] M. Dorigo, G. Di Caro. The Ant Colony Optimization Meta-heuristic. In D. Corne, M. Dorigo, F. Glover (eds), New Ideas in Optimization, pp. 11-21. McGraw-Hill, 1999

[4] M. Dorigo, V. Maniezzo, A. Colorni, The Ant System: Optimization by Colony of Cooperating Agents, IEEE Trans. on Systems, Man and Cybernetics, Part B, 26:1 (1996) 29-41

[5] M. Dorigo, L. Gambardella. Ant Colony System: A Cooperative Learning Approach to the Travelling Saleman Problem. IEEE Transactions on Evolutionary Computation 1 (1):53-66, 1997

[6] O. Cordón, F. Herrera, L. Moreno. Integración de Conceptos de Computación Evolutiva en un Nuevo Modelo de Colonias de Hormigas. En Actas de la CAEPIA'99. Seminario Especializado sobre Computación Evolutiva, Vol. II pp. 98-105. 1999.

[7] O. Cordón, I. Fernández de Viana, F. Herrera, L. Moreno. A New ACO Model Integrating Evolutionary Computation Concepts: The Best-Worst Ant System. Proc. of ANTS'2000. From Ant Colonies to Artificial Ants: Second International Workshop on Ant Algorithms, Brussels, Belgium, September 7-9, 2000, pp. 22-29

[8] Thomas Stützle and Holger H. Hoos MAX-MIN Ant System. Future Generation Computer Systems Journal 16(8):889-914, 2000

[9] B. Bullnheimer, G. Kotsis, C. Strauss, Parallelization Strategies for the Ant System. Parallel Problem Solving from Nature, 5th International Conference, Amsterdam, The Netherlands, September 27-30, 1998

[10] E.-G. Talbi, Olivier Roux, C. Fonlupt, D. Robillard, Parallel Ant Colonies for the quadratic assignment problem. Future Generation Computer Systems 17 (20001) pp. 441-449

[11] Thomas Stützle. Parallelization Strategies for Ant Colony Optimization.

[12] M. Middendorf, F. Reischle, H. Schmeck, Multi Colony Ant Algorithms.Journal of Heuristics Pap. tex. 15/04/2001

[13] F. Krüger, M. Middendorf, D. Merkle: Studies on a Parallel Ant System for the BSP Model ; Unpub. manuscript, 1998

[14] Z. Michalewicz. Genetic Algorithms + Data Structure = Evolution Programs. Springer-Verlag 1996

[15] E. Cantú-Paz. A survey of parallel genetic algorithms. Calculateurs Parallèles Réseaux of Systèmes Répartis, vol. 10 no.2 pp. 141-171, 1998

[16] G. Reinelt, TSLIB, pagina accesible en `http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/index.html`, última actualización 6 de Diciembre de 2001.

[17] C. Blum, A. Roli, Metaheuristics in Combinational Optimization: Overview and Conceptual Comparison. Technical Report 2001-13 Iridia, Free University of Brussels, 2001

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

[19] 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 , pp. 1013-1041. Vol 11, 1996

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

Actas del

II Congreso español sobre Metaheurísticas, Algoritmos evolutivos y bioinspirados

Gijón, 5 al 7 de febrero de 2003

maeb03

SESIÓN 2A: COMPUTACIÓN EVOLUTIVA Y TECNOLOGÍAS DEL SOFTWARE

SESIÓN 2B: CONTROL EVOLUTIVO

SESIÓN 3A: METAHEURÍSTICAS (I): OPTIMIZACIÓN