# Sistema de Hormigas para una Red Heterogénea de Computadoras

REPORTE TÉCNICO RT003-2001

Marta Almirón *malmiron@cnc.una.py*
Benjamín Barán *bbaran@cnc.una.py*

Centro Nacional de Computación
Universidad Nacional de Asunción
Casilla de Correos 1439
Campus Universitario de San Lorenzo
Paraguay

## Resumen

Sistema de hormigas *(Ant System)* es una nueva meta-heurística especialmente apropiada para problemas de optimización combinatoria. Se basa en el comportamiento estructurado de una colonia de hormigas donde individuos muy simples de una colonia se comunican entre sí por medio de una sustancia química denominada feromonas, estableciendo el camino más adecuado entre su nido y su fuente de alimentos. El método consiste en simular computacionalmente la comunicación indirecta que utilizan las hormigas para establecer el camino más corto guardando la información aprendida en una *matriz de feromonas*.

Considerando que las hormigas son entes básicamente autónomos que trabajan en paralelo con asincronismo implícito, el presente trabajo demuestra experimentalmente las ventajas de paralelizar esta técnica en un ambiente computacional asíncrono, utilizando una red de computadoras personales heterogéneas.

**Palabras Claves:**

Sistemas distribuidos y paralelismo, inteligencia artificial, optimización combinatoria.

## 1. Introducción

La observación de la naturaleza ha sido una de las principales fuentes de inspiración para la propuesta de nuevos paradigmas computacionales. Así nacieron diversas técnicas de Inteligencia Artificial como: los Algoritmos Genéticos (*Genetic Algorithms*), Templado Simulado (*Simulated Annealing*), Redes Neuronales (*Neural Networks*), y entre estas técnicas, el sistema basado en Colonias de Hormigas (*Ant System*) [DMC96].

Resulta realmente interesante como las hormigas buscan su alimento y logran establecer el camino más corto hasta llegar a él y regresar al nido. Al moverse una hormiga deposita una substancia química denominada feromonas como una señal odorífera para que las demás puedan seguirla. Las feromonas son un sistema de comunicación química entre animales de una misma especie, que transmiten información a través de señales odoríferas acerca del estado fisiológico, reproductivo y social, así como la edad, el sexo y el parentesco del animal emisor, las cuales son recibidas en el sistema olfatorio del animal receptor, quien interpreta esas señales, jugando un papel importante en la organización y la supervivencia de muchas especies [DMC96].

En principio, una hormiga se mueve esencialmente al azar pero las demás deciden con buena probabilidad seguir el camino con mayor cantidad de feromonas. Considere la Figura 1 en donde se observa como las hormigas establecen el camino más corto. En la figura (a) las hormigas llegan a un punto donde tienen que decidir por uno de los caminos que se les presenta, lo que resuelven de manera aleatoria, la mitad de las hormigas de dirigirán hacia un extremo y la otra mitad hacia el otro extremo como ilustra la figura (b). Como las hormigas se mueven aproximadamente a una velocidad constante, las que eligieron el camino más corto alcanzarán el otro extremo más rápido que las que tomaron el camino más largo, depositando mayor cantidad de feromona por unidad de longitud, como ilustra la figura (c). La mayor densidad de feromonas depositadas en el trayecto más corto hace que éste sea más deseable para las siguientes hormigas y por lo tanto la

mayoría elige transitar por él. Considerando que la evaporación de la sustancia química hace que los caminos menos transitados sean cada vez menos deseables y la realimentación positiva del camino con más feromonas, resulta claro que al cabo de un tiempo casi todas las hormigas transiten por el camino más corto.

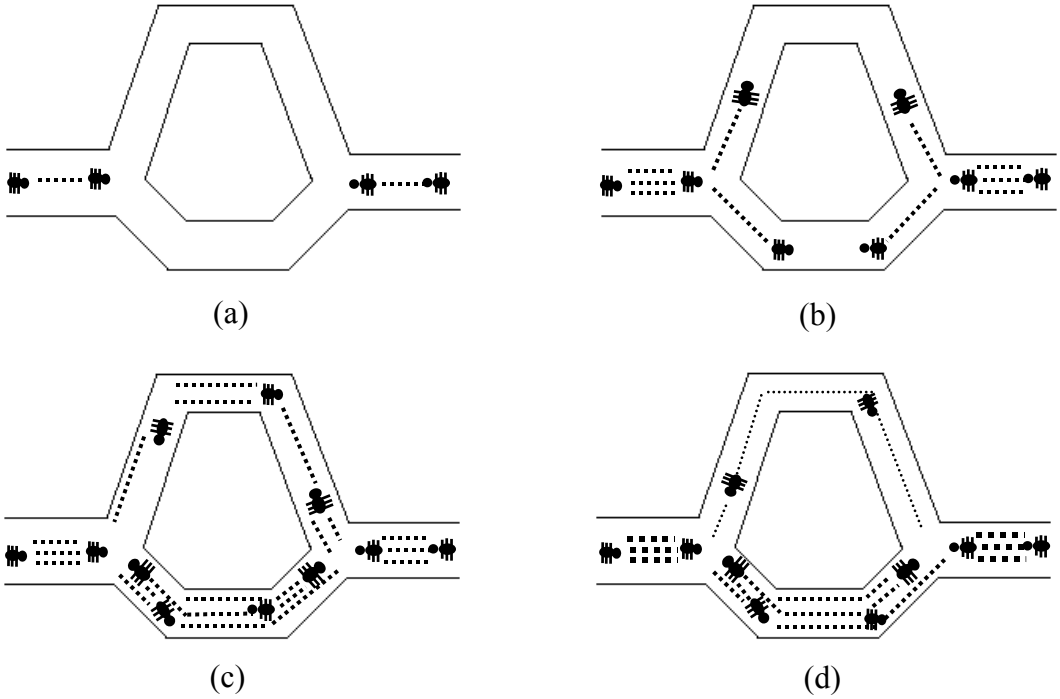


**Figura 1:** Comportamiento de las hormigas reales.

En base al comportamiento descrito de las hormigas, Dorigo et al. [DMC96] propuso una técnica conocida como *Ant system* para el conocido paradigma del cajero viajante (TSP – *Traveler Salesman Problem*) [Rei00] que presentaba tres variantes que se diferencian simplemente en el momento y la manera de actualizar una matriz que representa las feromonas de los sistemas biológicos:

- *Ant-density*: con actualización constante de las feromonas por donde pasa una hormiga.
- *Ant-quantity*: con actualización de feromonas inversamente proporcional a la distancia entre 2 ciudades recorridas.
- *Ant-cycle*: con actualización de feromonas inversamente proporcional al trayecto completo, al terminar un recorrido. Este último presentó mejores resultados y las siguientes investigaciones se centraron en él.

Recientemente, Dorigo y Gambardella trabajaron en varias versiones extendidas del paradigma *AS*. *Ant-Q* [GD95, DG96] es un híbrido entre *AS* y *Q-learning*, un conocido algoritmo de aprendizaje con realimentación positiva [GD95, DG96]. *Ant Colony System (ACS)* es una extensión de *Ant-Q* publicado en 1997 [DG97], en el que se presentaron mejoras del algoritmo en tres aspectos principales: i) la regla de transición de estados, con la que se ofrece un balance entre la exploración de nuevos caminos y explotación a priori del conocimiento acumulado acerca del problema, ii) la regla de actualización global que permite actualizar la matriz de feromonas solo con el mejor tour encontrado hasta el momento iii) la regla de actualización local que permite a todas las hormigas actualizar la matriz de feromonas al terminar su tour. En dicha publicación se aplica además búsqueda local, utilizando el conocido método 3-opt [JM97] para el problema del cajero viajante, que intenta reducir la longitud encontrada intercambiando los arcos. En particular, este método realiza tres cortes en el tour encontrado e intercambia las ciudades destino, evitando de este modo invertir el sentido de las ciudades visitadas.

Otra variante de *AS*, conocida como *Max-Min Ant System (MMAS)*, fue presentada por Tomas Stützle y Holger Hoos [SH96], permitiendo actualizar la matriz de feromonas, solo a la hormiga con el mejor tour. Esto acelera la convergencia, pero puede llevar a estancamientos en soluciones sub-óptimas. A fin de evitar convergencias prematuras, se

propuso poner un límite máximo y otro mínimo dentro de los cuales puede variar la cantidad de feromonas [SH96]. La aplicación de búsqueda local mejora notoriamente los resultados experimentales [SH97].

Como *Ant System* es una clase de algoritmo distribuido que consiste generalmente en un conjunto de numerosos agentes cooperativos muy simples que intercambian información de manera indirecta, el paralelismo es inherente al funcionamiento del algoritmo, es decir, el comportamiento de una hormiga es independiente de todas las demás hormigas durante la misma iteración. Se propusieron varias estrategias de paralelización. En [BKS97] fue publicada una implementación paralela síncrona y otra parcialmente asíncrona que se resumen a continuación.

a) En la implementación paralela síncrona, un proceso inicial (master) levanta a un conjunto de procesos hijos, uno para cada hormiga. Después de distribuir la información inicial acerca del problema, cada proceso inicia la construcción del camino y calcula la longitud del tour encontrado. Después de terminar este procedimiento, los resultados son enviados al master, quien se encarga de actualizar el nivel de feromonas y calcular el mejor tour encontrado hasta ahora. Se inicia una nueva iteración con el envío de una nueva matriz de feromonas.

b) En la implementación parcialmente asíncrona, se propone reducir la frecuencia de la comunicación. Para esto, cada hormiga realiza un cierto número de iteraciones del algoritmo secuencial, independientemente de las otras hormigas. Solo después de estas iteraciones locales, se realiza una sincronización global entre las hormigas. Entonces el master actualiza el nivel de feromonas y se inicia el proceso de nuevo.

Como una extensión de éste trabajo, Tomas Stützle presenta otra estrategia de paralelización [Stü98], proponiendo corridas independientes y paralelas de un mismo algoritmo, (ACO o MMAS), evitando de éste modo el *overhead* de comunicación. Cabe destacar que en este modelo de paralelización no existe comunicación entre los procesos que corren en diferentes máquinas. El método funciona solo si el fundamento del algoritmo es aleatorio como en el caso de *Ant System*. La mejor solución de k corridas es elegida al final. En caso de aplicar búsqueda local, Stützle propone dos estrategias pero sin presentar resultados experimentales:

a) Una implementación síncrona al tener un proceso master que es utilizado para actualizar las estructuras de datos y construir soluciones iniciales que son enviadas a los procesadores hijos que se encargan de aplicar búsqueda local sobre ellos. El master recoge las soluciones óptimas locales y de encontrarse un número suficiente de tales soluciones se actualiza la matriz de feromonas antes de construir más soluciones.

b) Una implementación asíncrona en la que un procesador guarda la matriz de feromonas y la actualiza, mientras que uno o varios procesadores pueden usar la matriz de feromonas para construir soluciones y enviar a aquellos procesadores que aplicaran búsqueda local sobre éstos resultados.

Los autores del presente trabajo presentaron otra estrategia de paralelización [BA00], proponiendo que un procesador master levante a los demás procesos. Cada procesador hijo realiza la computación del problema para un cierto número de hormigas, obteniendo resultados parciales que serán transmitidos a los otros procesadores, colaborando todos en la solución del problema global. Los resultados fueron obtenidos en una red de computadoras personales con hasta 6 procesadores.

En los últimos años fueron probadas numerosas implementaciones de los algoritmos *AS* con el objeto de resolver problemas tan diversos como:

- el paradigma del cajero viajante *(Traveling Salesman Problem*) [DMC96, DG97];
- el problema de ruteo de vehículos *(Vehicle Routing Problem)* [BHS99];
- el problema del ordenamiento secuencial *(Sequential Ordering Problem)* [GD97];
- redes de telecomunicaciones *(Telecommunications Networks)* [CD98, BS00];
- el problema de asignación cuadrática *(Quadratic Assignment Problem)* [MC99].

El presente trabajo propone paralelizar el algoritmo *Ant System (AS)* en un ambiente totalmente asíncrono y heterogéneo en una red de 12 computadoras personales y realizar corridas experimentales con el problema simétrico del cajero viajante (TSP) para 100 ciudades [Rei00]. De esta manera tenemos que, dadas $N$ ciudades y las distancias entre estas (idénticas en ambos sentidos), se desea encontrar el camino más corto que permita recorrer las $N$ ciudades, regresando al punto de partida, sin pasar por una misma ciudad más de una vez. Para esto, agentes computacionales muy simples llamados *hormigas* trabajan en cada procesador de una red de computadoras, explorando el espacio de soluciones, comunicando en

forma asíncrona a los demás procesadores los resultados más prometedores, consolidando la información recogida en *matrices de feromonas* propias de cada procesador, que servirán para guiar la búsqueda de mejores soluciones en cada uno de los procesadores de la red, sin necesidad de sincronizar los procesos. Con esto, cada procesador puede contar con su propia matriz de feromonas, posiblemente diferente a las matrices utilizadas en otros procesadores, que guiarán las búsquedas de buenas soluciones de forma posiblemente diferenciada para cada uno de los procesadores que colaboran en la búsqueda de las soluciones globales, proporcionando una diversidad que podrá ser beneficiosa (como ocurre con los algoritmos genéticos) y que en general, no puede aprovecharse en las implementaciones puramente secuenciales.

La siguiente sección presenta el algoritmo *Ant System* en una versión secuencial simplificada, mientras la técnica de paralelización propuesta se explica en la sección 3. Los resultados experimentales quedan para la sección 4. Finalmente, se presentan las conclusiones en la sección 5.

## 2. Sistema de Hormigas

Esta novedosa técnica se inspira en el comportamiento de las hormigas, animales casi ciegos pero con la habilidad de optimizar el camino hasta llegar a la fuente de su alimento y regresar al nido. El algoritmo utiliza simples agentes llamados *hormigas* con las siguientes características:

- cada hormiga elige la siguiente ciudad a ser visitada, en forma aleatoria, calculando una probabilidad que es una función de la distancia entre las ciudades origen y destino y la cantidad de feromona presente en el arco que las conecta;
- se obliga a cada hormiga a realizar un tour legal, esto es, visitar cada ciudad una sola vez. Viajar a una ciudad ya visitada no está permitido hasta que complete todo el viaje (esto es controlado por una lista tabú);
- cuando una hormiga viaja de la ciudad *i* a la ciudad *j*, deposita una cantidad de feromona, en el arco *(ij)*, marcando el camino recorrido.

Estos agentes se diferencian de las hormigas reales en ciertos aspectos como:

- tienen cierta capacidad de memoria (ej., para guardar la lista tabú),
- no son completamente ciegas y
- viven en un ambiente de tiempo discreto.

Para el presente trabajo, se considera un conjunto de *MAXC* ciudades que deben ser visitadas una sola vez con el objeto de encontrar la longitud mínima de recorrido y se define $b_i(t)$ *(i=1, ...,MAXC)* como el número de hormigas en la ciudad *i* al tiempo *t*. Por consiguiente, el número total de hormigas *MAXH* estará dado por:

$$\boldsymbol{MAXH} = \sum_{i=1}^{MAXC} b_i(t) \qquad (1)$$

Para satisfacer la restricción de que una hormiga visite todas las ciudades una sola vez, se asocia a cada *hormiga k* una estructura de datos llamada lista tabú, $tabu_k$, que guarda las ciudades ya visitadas por dicha hormiga. Una vez que todas las ciudades hayan sido recorridas, el trayecto o tour (ciclo) es completado, la lista tabú se vacía y nuevamente la hormiga está libre para iniciar un nuevo tour. Se define como $tabu_k(s)$ al elemento s-esimo de la lista tabú de la *hormiga k*.

Dado un conjunto de *MAXC* ciudades, denominamos $d_{ij}$ a la longitud del camino entre las ciudades *i*, *j*; en el caso del Problema de Cajero Viajante Euclidiano. El punto de partida para la solución del problema simétrico del cajero viajante, es la matriz de distancias $D = \{d_{ij}$ , distancia entre la ciudad *i* y la ciudad *j*$\}$, a partir de la cual se calcula la visibilidad $\eta_{ij} = 1/d_{ij}$ . Por su parte, se denota como $\tau = \{\tau_{ij}\}$ a la matriz de feromonas a ser utilizada para consolidar la información que va siendo recogida por las hormigas; en otras palabras, la cantidad de feromona que se va almacenando entre cada par de ciudades *(i j)*.

$\tau_{ij}(t)$ denota la intensidad de las feromonas del arco *(i j)* en el tiempo *t*, y se actualiza según:

$$\tau_{ij}(t+1) = \rho \times \tau_{ij}(t) + \Delta\tau_{ij}\left(t, t+1\right) \qquad (2)$$

donde ρ es el coeficiente de persistencia de las feromonas, de forma tal que (1-ρ) representa la evaporación de la substancia entre *t* y *t+1*, mientras que la cantidad de feromona depositada en un arco *(i j)*, en dicho intervalo de tiempo, está dada por:

$$\Delta\tau_{ij}(t, t+1) = \sum_{k=1}^{MAXH} \Delta\tau_{ij}^{k}(t, t+1) \qquad (3)$$

con $\Delta\tau_{ij}^{k}$ (t, t+1) representando la cantidad de feromona depositada en el arco *(i j)* por la hormiga *k-esima* entre *t* y *t+1*.

Durante la ejecución del algoritmo *Ant System*, cada *hormiga* elige en forma probabilística la próxima ciudad a visitar, realizando un cálculo de probabilidad que es función de la distancia y la cantidad de feromona depositada en el arco que une a las ciudades origen-destino, esto es:

$$p_{ij}(t) = \begin{cases} \dfrac{\left[\tau_{ij}(t)\right]^{\alpha} \times \left[\eta_{ij}\right]^{\beta}}{\sum_{j \notin Tabu_k} \left[\tau_{ij}(t)\right]^{\alpha} \times \left[\eta_{ij}\right]^{\beta}} & \text{si } j \notin \text{Tabu}_k \\ 0 & \text{de otra manera} \end{cases} \qquad (4)$$

donde α y β son constantes que expresan la importancia relativa del sendero de feromonas y la distancia entre las ciudades respectivamente. Así, un alto valor de α significa que el sendero de feromonas es muy importante y que las hormigas tienden a elegir caminos por los cuales otras hormigas ya pasaron. Si por el contrario, el valor de β es muy alto, una hormiga tiende a elegir la ciudad más cercana.

En el instante t, las hormigas se mueven de una ciudad a la siguiente (movimiento conocido como: *iteración*), en donde se encontrarán en el instante *t+1*. Lógicamente, al cabo de *(MAXC – 1)* iteraciones, las hormigas han visitado la última ciudad y están en condiciones de regresar a su ciudad origen, posiblemente para actualizar la matriz de feromonas con la información recogida en el tour completo.

El proceso se repite iterativamente hasta que se cumpla algún criterio de parada. En este trabajo, el proceso termina si el contador de tour alcanza un número máximo de ciclos *NCMAX* (definido por el usuario) o todas las hormigas realizan el mismo tour. En este último caso, es evidente que las hormigas han dejado de buscar nuevas soluciones, lo que constituye un criterio de convergencia del algoritmo (similar a la uniformización de la población de un algoritmo genético [BCC98]).

La cantidad de feromonas depositada en el trayecto es proporcional a la distancia del tour completo encontrado y por lo tanto, es de esperar una apreciable capacidad de búsqueda de soluciones globales. Para esto, se realiza el siguiente cálculo de $\Delta\tau_{i,j}^{k}$:

$$\Delta\tau_{ij}^{k} = \begin{cases} \dfrac{Q}{L_k} & \text{si la hormiga } k\text{-}esima \text{ camina por el arco } (i, j) \\ 0 & \text{de otra manera} \end{cases} \qquad \textbf{(5)}$$

donde $Q$ es una constante y $L_k$ es la longitud del tour completo realizado por la hormiga *k*.

El Pseudocódigo 1 se presenta el algoritmo secuencial de *Ant System* en su versión más utilizada [DMC96].

**Pseudocódigo 1:** Algoritmo *Ant System*

1.Fase de inicialización
Inicializar contador de ciclos ***NC***
Para cada arco ***(i.j)***:
valor inicial de $\tau_{ij}(t) = c$ /**c*= constante positiva pequeña*/
$\Delta\tau(ij) = 0$
Colocar ***MAXH*** hormigas en ***N*** ciudades

2.Colocar la ciudad origen en lista $tabu_k$ de cada hormiga

3.Repetir hasta llenar $tabu_k$
Para cada hormiga:
Elegir próxima ciudad a ser visitada según ecuación (4)
Mover la hormiga a la próxima ciudad
Insertar la ciudad en $tabu_k$

4.Repetir para cada hormiga ***k***
Regresar a la ciudad origen
Calcular la longitud $L_k$ del ciclo
Guardar el camino más corto hasta ciclo **NC**: $L_0^{NC} = min\{L_0^{NC-1}; min_k\{L_k\}\}$
Para cada arco ***(i,j)***
Calcular $\Delta\tau_{ij}$ según ecuación (3)

5.Para cada arco ***(i,j)***
Actualizar $\tau_{ij}$ según ecuación (2)
$\Delta\tau_{ij} = 0$

6.Aumentar contador de ciclos ***NC***
Si ***NC*** < $NC_{max}$
Vaciar $tabu_k$
Ir a la fase 2
Sino
Imprimir camino más corto $L_0^{NC}$
Fin

## 3. Estrategia de paralelización implementada

*Ant System* es un algoritmo inspirado en la observación de la naturaleza, y en particular, en una colonia de hormigas en la que cada individuo aporta su trabajo individual para alcanzar el objetivo común, la supervivencia.

La colonia entera trabaja como un equipo en perfecta coordinación aunque cada hormiga hace su trabajo en forma independiente, tomando sus propias decisiones en base a su ambiente localizado (ej., feromona percibida). Cada individuo comunica indirectamente a los demás su experiencia a través de la sustancia química denomina feromonas. Análogamente, el algoritmo utiliza agentes computacionales asíncronos y paralelos, llamados *hormigas*. Cada uno de estos agentes va construyendo su tour con la única restricción de no viajar a una ciudad ya visitada con anterioridad y dejando en la matriz de feromonas un rastro por los caminos ya transitados, sin una comunicación directa con las demás hormigas. Es evidente que el paralelismo está implícito en el mismo algoritmo. Consecuentemente, el presente trabajo propone que cada

procesador realice la computación del problema para cierto número de hormigas, obteniendo resultados parciales que serán trasmitidos a los otros procesadores (sin ninguna sincronización), colaborando todos en la solución del problema global.

Por su parte, el asincronismo aquí propuesto elimina los tiempos muertos producidos por la espera en la sincronización de la comunicación. En consecuencia, se realizan corridas independientes del algoritmo en cada procesador hijo, teniendo un master que comunique los parámetros y el número de ciclos que deben realizar cada uno de ellos, éstos a su vez le comunican al master cuando han terminado (Figura 2). Entre los procesadores hijos las hormigas que hayan obtenido mejores resultados migran de un computador a otro respetando políticas migratorias semejantes a los algoritmos genéticos paralelos [BA00]. Cada proceso recibe a las hormigas migrantes, éstas se ubican en las ciudades de origen, donde se aplica un criterio de selección a fin de mantener el número original de hormigas por cada ciudad, las hormigas que hayan encontrado mejores soluciones, tienen mayor probabilidad de sobrevivir a la selección.

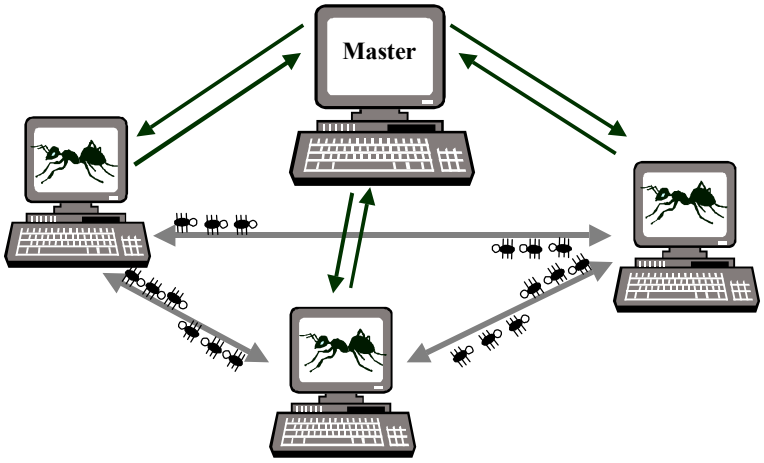


**Figura 2:** Estrategia de paralelización.

A continuación se presenta la versión paralela implementada que utiliza un proceso *Master* que se encarga de administrar la implementación paralela (Pseudocódigo 2) incluyendo el lanzamiento de los procesos *Esclavos* (Pseudocódigo 3), para que éstos realicen los cálculos propiamente dichos.

1. Fase de inicialización
   Levantar procesos
   Enviar parámetros a cada esclavo
   Fin= Falso

2. Repetir mientras no sea Fin
   Recibir solución de los esclavos
   Guardar mejor solución
   Si todos los procesos terminaron
   Fin= Verdadero

3. Eliminar procesos esclavos

**Pseudocódigo 2:** Proceso Master.

1. Fase de inicialización
   Recibir parámetros
   Inicializar variables

2. Repetir mientras (*NC* < *NCMAX* y todas las
   hormigas no realicen el mismo tour)
   Mover hormigas hasta completar tour
   Calcular distancia recorrida para cada hormiga
   Escoger hormigas para emigrar
   Enviar hormigas emigrantes a otros procesos
   Recibir hormigas migrantes de otros procesos
   Ubicar migrantes en su ciudad origen
   Seleccionar las hormigas que sobrevivirán
   Actualizar matriz de feromonas
   Guardar camino más corto

3. Enviar mejor solución al master.

**Pseudocódigo 3:** Proceso Esclavo.

## 4. Resultados experimentales

Los experimentos fueron realizados en una red local *Ethernet* de doce computadoras personales, 8 con procesadores AMD-K6 de 350 MHz, 128 MB de memoria RAM y 4 computadoras personales con procesadores AMD-K6 de 450 MHz y 64 MB de memoria RAM, corriendo bajo el sistema operativo Linux Mandrake 7.0. Las implementaciones fueron codificadas en lenguaje C utilizando librerías de PVM *(Parallel Virtual Machine)*.

El problema KroA100 [Rei00] para 100 ciudades fue utilizado con el propósito de probar el desempeño del algoritmo en un ambiente que requiera de un importante uso de recursos computacionales. Conforme ilustra la Tabla 1, se hicieron experimentos con 1, 2, 4, 6, 8, 10 y 12 computadore s heterogéneos en red.

| Nº de Proc. | Tiempo | Distancia | Rango |
|---|---|---|---|
| 1 | 6468 | 22489.7871 | 15 |
| 1 | 6475 | 22457.4141 | 14 |
| 1 | 6453 | 22477.3262 | 14 |
| 1 | 6425 | 22434.3047 | 13 |
| 1 | 6377 | 22477.3203 | 13 |
| 1 | 6305 | 22387.5996 | 12 |
| 1 | 6298 | 22489.3379 | 12 |
| 1 | 6306 | 22367.1152 | 11 |
| 1 | 6163 | 22457.4121 | 11 |
| 1 | 6287 | 22387.5957 | 11 |
| 2 | 3380 | 22347.07422 | 10 |
| 2 | 3377 | 22347.0742 | 9 |
| 2 | 3345 | 22639.6758 | 9 |
| 2 | 3360 | 22512.2422 | 9 |
| 2 | 3373 | 22402.0234 | 9 |
| 2 | 3375 | 22347.0742 | 8 |
| 2 | 3371 | 22374.4375 | 8 |
| 2 | 3353 | 22347.0723 | 7 |
| 2 | 3339 | 22347.0703 | 6 |
| 2 | 3349 | 22347.0664 | 6 |
| 4 | 1695 | 22489.3379 | 8 |
| 4 | 1702 | 22457.416 | 6 |
| 4 | 1707 | 22403.4063 | 6 |
| 4 | 1683 | 22430.0352 | 5 |
| 4 | 1694 | 22347.0664 | 5 |
| 4 | 1683 | 22347.0723 | 4 |
| 4 | 1684 | 22347.0664 | 4 |
| 4 | 1683 | 22347.0664 | 3 |
| 4 | 1684 | 22330.7773 | 3 |
| 4 | 1682 | 22254.9981 | 1 |
| 6 | 1201 | 22529.9141 | 7 |
| 6 | 1202 | 22478.1543 | 7 |
| 6 | 1201 | 22474.7051 | 6 |
| 6 | 1201 | 22447.6328 | 5 |
| 6 | 1123 | 22489.1758 | 4 |

| Nº de Proc. | Tiempo | Distancia | Rango |
|---|---|---|---|
| 6 | 1201 | 22445.6191 | 4 |
| 6 | 1132 | 22430.0332 | 3 |
| 6 | 1133 | 22422.5 | 3 |
| 6 | 1201 | 22365.125 | 3 |
| 6 | 1202 | 22323.166 | 2 |
| 8 | 902 | 22729.6133 | 8 |
| 8 | 902 | 22693.5488 | 7 |
| 8 | 902 | 22652.7832 | 6 |
| 8 | 902 | 22650.084 | 5 |
| 8 | 901 | 22603.7832 | 4 |
| 8 | 902 | 22599.8906 | 4 |
| 8 | 901 | 22556.2871 | 3 |
| 8 | 902 | 22436.0879 | 3 |
| 8 | 902 | 22350.1328 | 2 |
| 8 | 901 | 22285.1895 | 1 |
| 10 | 773 | 22776.8867 | 7 |
| 10 | 705 | 22980.666 | 6 |
| 10 | 750 | 22663.1387 | 6 |
| 10 | 712 | 22656.6055 | 5 |
| 10 | 707 | 22611.8848 | 4 |
| 10 | 647 | 22602.5117 | 3 |
| 10 | 717 | 22587.2266 | 3 |
| 10 | 713 | 22494.1367 | 2 |
| 10 | 714 | 22439.3145 | 2 |
| 10 | 735 | 22404.2676 | 1 |
| 12 | 666 | 23032.4258 | 6 |
| 12 | 608 | 22822.9727 | 5 |
| 12 | 608 | 22790.6465 | 4 |
| 12 | 611 | 22726.4668 | 4 |
| 12 | 608 | 22726.5137 | 3 |
| 12 | 614 | 22674.9844 | 3 |
| 12 | 605 | 22647.9414 | 2 |
| 12 | 610 | 22585.0664 | 2 |
| 12 | 550 | 22534.4785 | 1 |
| 12 | 612 | 22438.5313 | 1 |

**Tabla 1:** Resultados obtenidos para el problema KroA100.

Los resultados experimentales obtenidos con 70 corridas (10 corridas por cada número de procesadores) son mostrados en la Tabla 1, donde se pueden apreciar los tiempos de cada corrida y la mejor solución (distancia mínima) encontrado en cada corrida. La última columna de esta tabla muestra el *Rango* de cada corrida con respecto al total de 70 corridas. Una corrida *i* es de rango 1 si no existe otra corrida *j* para la cual se obtiene una mejor solución en un tiempo menor o igual al requerido por la corrida *i*. En otras palabras, una solución *i* es de rango 1 solo si es *Pareto óptima* con respecto a todo el universo de corridas, considerando la minimización simultanea del tiempo y la distancia del tour. En general, para encontrar las corridas con rangos $k > 1$, se elimina del universo todas las corridas con rangos menores a *k* y en este conjunto restante, se encuentran todas las corridas *Pareto óptimas*; es decir, con soluciones que no pueden ser mejoradas sin utilizar un mayor tiempo de cómputo. En resumen, el algoritmo utilizado para encontrar el rango de las corridas es el siguiente:

**Pseudocódigo 4:** Cálculo del rango de las corridas

1. Inicialización: Rango = 1, Universo = {conjunto de todas las corridas}

2. Para cada corrida *i* del Universo, hacer
   Comparar con todas las corridas $j \neq i$
   Si la corrida *i* es mejor (menor tour y menor o igual tiempo) que la corrida *j*,
   marcar *j* como *dominada*
   Si la corrida *j* es mejor (menor tour y menor o igual tiempo) que la corrida *i*,
   marcar *j* como *dominada*

3. Asignar Rango a todas las corridas no *dominadas* (soluciones Pareto óptimas, considerando tiempo y distancia)

4. Universo = {conjunto de todas las corridas *dominadas*}

5. Rango = Rango + 1

6. Si el Universo no está vacío, ir a 2

Claramente, las soluciones de rango *k* son mejores que las de rango *(k+1)* (pues estas son dominadas por las primeras) por lo que se propone utilizar el rango promedio para cada número de procesadores como una forma de determinar si existe una ventaja en utilizar varios procesadores en paralelo. En efecto, la Tabla 2 presenta en la primera columna el número de procesadores utilizados en cada caso, en la segunda columna la sumatoria de los rangos alcanzados en 10 corridas típicas, la tercera columna presenta el promedio de los rangos y por último, en la cuarta columna la desviación estándar para cada uno de los casos. Es fácil notar una clara correlación inversa entre el número de procesadores y el promedio de rangos ($\rho < -0{,}8$ ), es decir, la calidad de las corridas considerando distancia del tour y tiempo de ejecución mejora con el número de procesadores, de lo que resulta obvia la ventaja de paralelizar el método de las colonias de hormigas. Sin embargo, se puede notar que la ventaja de utilizar paralelismo está verificada experimentalmente solo en promedio, dado que el método es probabilístico y no determinístico. En efecto, la Tabla 1 muestra por ejemplo una solución óptima (de rango 1) utilizando 4 procesadores, mientras que existen soluciones de rango 2 utilizando 12 procesadores, lo que no impide que en promedio, el rango utilizando 4 procesadores sea superior que utilizando 12 procesadores (ver Tabla 2).

En conclusión, se comprueba experimentalmente que la utilización de un número creciente de procesadores (hasta 12 procesadores en nuestros experimentos) mejora la calidad de una corrida, considerando simultáneamente la distancia del tour obtenido y el tiempo de ejecución. De lo realizado se puede conjeturar que en el caso de las colonias de hormigas, el número de procesadores puede ir creciendo considerablemente, obteniéndose mejores corridas (en promedio) en la medida que crece el número de procesadores.

| Número de Procesadores | Suma de los Rangos | Promedio de los Rangos | Desviación Estándar |
|---|---|---|---|
| 1 | 126 | 12.6 | 1.43 |
| 2 | 81 | 8.1 | 1.37 |
| 4 | 45 | 4.5 | 1.96 |
| 6 | 44 | 4.4 | 1.78 |
| 8 | 43 | 4.3 | 2.21 |
| 10 | 39 | 3.9 | 2.02 |
| 12 | 31 | 3.1 | 1.66 |

**Tabla 2:** Promedios y Desviación Estándar de los rangos

## 5. Conclusiones

*Ant System* es un nuevo algoritmo meta-heurístico para resolver problemas de optimización combinatoria, especialmente adecuado para su paralelización asíncrona en ambientes distribuidos, como el de las redes de computadoras. Para demostrar el desempeño del algoritmo paralelo propuesto en este trabajo, fue utilizado el paradigma del cajero viajante para 100 ciudades [Rei00], con un número creciente de hasta 12 computadores heterogéneos conectados en una red de área local.

Las pruebas experimentales demuestran un buen desempeño del método con un número creciente de computadores, lo que nos permite postular su utilización con problemas de mayor tamaño y complejidad, aprovechando un número creciente de computadoras disponibles en las redes actuales. Como extensión del presente trabajo se espera realizar experimentos paralelos del método presentado con otros tipos de problemas tales como el problema de la asignación cuadrática *(QAP - Quadratic Assignment Problem)* y el problema del ruteo de vehículos *(VRP - Vehicle Routing Problem)*. Finalmente, se espera verificar que el método paralelo asíncrono propuesto puede inclusive ser aprovechado en grandes redes de alta heterogeneidad, como Internet.

## Bibliografía


[BA00] Barán B. y Almirón M. "Colonias distribuidas de hormigas en un entorno paralelo asíncrono". *XXVI Conferencia Latinoamericana de Informática – CLEI'00*, México, 2000.

[BCC98] Barán B., Chaparro E. y Cáceres N., "A-Teams en la Optimización del Caudal Turbinado de una Represa Hidroeléctrica", *Conferencia Iberoamericana de Inteligencia. Artificial – IBERAMIA'98*, Portugal, 1998.

[BHS99] Bullnheimer B., Hartl R. y Strauss C. "An improved ant system algorithm for the vehicle routing problem". *Annals of Operations Research* (Dawind, Feichtinger and Hartl (eds.): Nonlinear Economic Dynamics and Control, 1999.

[BKS97] Bullnheimer B., Kotsis G. y Strauss C. "Parallelization Strategies for the Ant System", Reporte Técnico POM 9/97, Universidad de Viena, Viena – Austria, 1997.

[BS00] Barán B. y Sosa R., "A New approach for AntNet routing" *Ninth International Conference on Computer Commnunications and Networks*. Las Vegas, Nevada. 2000.

[CD98] Di Caro G. y Dorigo M., "Mobile Agents for Adaptive Routing", *Proceedings of the 31st Hawaii International Conference on System*, Hawaii, 1998.

[DG96] Dorigo M. y Gambardella L., "A study of some properties of Ant-Q". *IV International Conference on Parallel Problem from Nature*, Berlin – Alemania: Springer-Verlag, pp. 656-665, 1996.

[DG97] Dorigo M. y Gambardella L., "Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem". *IEEE Transaction on Evolutionary Computation*, Vol 1, No. 1, pp. 53-66, 1997.

[DMC96] Dorigo M., Maniezzo V. y Colorni A., "The Ant System: Optimization by a colony of cooperating agents", *IEEE Transaction on Systems, Man, & Cybernetics*, Vol 26, No. 1, pp. 1-13, 1996.

[Dor98] Dorigo M., *Ant Colony System*, URL: http://iridia.ulb.ac.be/dorigo/ACO/ACO.html.

[GD95] Gambardella L. y Dorigo M., "Ant-Q: A reinforcement learning approach to the traveling salesman problem" *12th International Conference on Machine Learning*. San Francisco, pp. 252-260, 1995.

[GD97] Gambardella L. y Dorigo M., "HAS-SOP: An Hybrid Ant System for the Sequential Ordering Problem", Reporte Técnico *IDSIA11-97*, Lugano-Suiza, 1997.

[JM97] Johnson D.S. y McGeoch L.A., "The traveling salesman problem: a case study in local optimization", *Local Search in Combinatorial Optimization*, Eds. New York: Wiley: New York, 1997.

[MC99] Maniezzo V. y Colorni A. "The Ant System applied to the Quadratic Assignment Problem*". IEEE Transactions on Knowdledge and Data Engineering*, 1999.

[SH96] Stützle T. y Hoos H., "Improvements on the Ant System: Introducing Max-Min Ant System". *International Conference on Artificial Neural Networks and Genetic Algorithms*, Viena – Austria, 1997.

[SH97] Stützle T. y Hoos H., "Max-Min Ant System and Local Search for Combinatorial Optimization Problems". *2nd International Conference on Metaheuristics* – MIC97, Francia, 1997.

[Stü98] Stützle T., "Parallelization Strategies for Ant Colony Optimization", *Proceedings of Parallel Problem Solving from Nature* – PPSN-V, Springer Verlag, Vol. 1498, pp. 722-731, 1998.

[Rei00] Reinelt G., "TSP", http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/TSPLIB.html. Universität Heidelberg, Institut für Angewandte Mathematik, Germany. 2000.