Colonia de Hormigas en un Ambiente Paralelo Asíncrono

Benjamín Barán
Universidad Nacional de Asunción
Centro Nacional de Computación
San Lorenzo, Paraguay
bbaran@cnc.una.py

y

Marta Almirón
Universidad Nacional de Asunción
Centro Nacional de Computación
San Lorenzo, Paraguay
malmiron@cnc.una.py

Abstract

Ant Colony System (ACS) studies ant artificial systems that take inspiration from the collective behavior of real ants to solve combinatorial optimization problems.

ACS is based on the structured behavior of ant colony, where very simple individuals communicate information to each other using a chemical substance denominated pheromone, establishing the shortest paths from their nest to a feeding sources and back. The method consists in a computational simulation of the indirect communication that uses agents called ants to establish the shortest path keeping the information learned in a matrix of pheromone.

Considering the ants are basically independent agents working in parallel without synchronization, this paper proposes a parallel asynchronous implementation in a network of personal computers, presenting experimental results that prove the usefulness of parallelism and the viability of the proposed asynchronous implementation.

Keywords: Parallel and Distributed Systems, Artificial Intelligence, Combinatorial Optimization.

Resumen

Ant Colony System (ACS) estudia los sistemas artificiales de hormigas inspirados en la conducta colectiva de hormigas reales, utilizados para resolver problemas de optimización combinatoria.

ACS 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 feromona, 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 agentes básicamente autónomos que trabajan en paralelo, este trabajo presenta una implementación paralela asíncrona del algoritmo ACS en una red de computadoras personales, presentando resultados experimentales que prueban la ventaja de usar paralelismo y la viabilidad de la implementación propuesta.

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 Colony System*) [1].

Resulta realmente interesante analizar como las hormigas buscan su alimento y logran establecer el camino más corto para luego regresar a su nido. Para esto, al moverse una hormiga, deposita una sustancia química denominada *feromona* como una señal odorífera para que las demás puedan seguirla. Las feromonas son un sistema indirecto de comunicación química entre animales de una misma especie, que transmiten información 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 olfativo del animal receptor, quien interpreta esas señales, jugando un papel importante en la organización y la supervivencia de muchas especies [2].

Al iniciar la búsqueda de alimento, una hormiga aislada se mueve a ciegas, es decir, sin ninguna señal que pueda guiarla, pero las que le siguen 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. En consecuencia, 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, quedando depositado 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 en el camino con más feromonas, resulta claro que al cabo de un tiempo casi todas las hormigas transiten por el camino más corto.

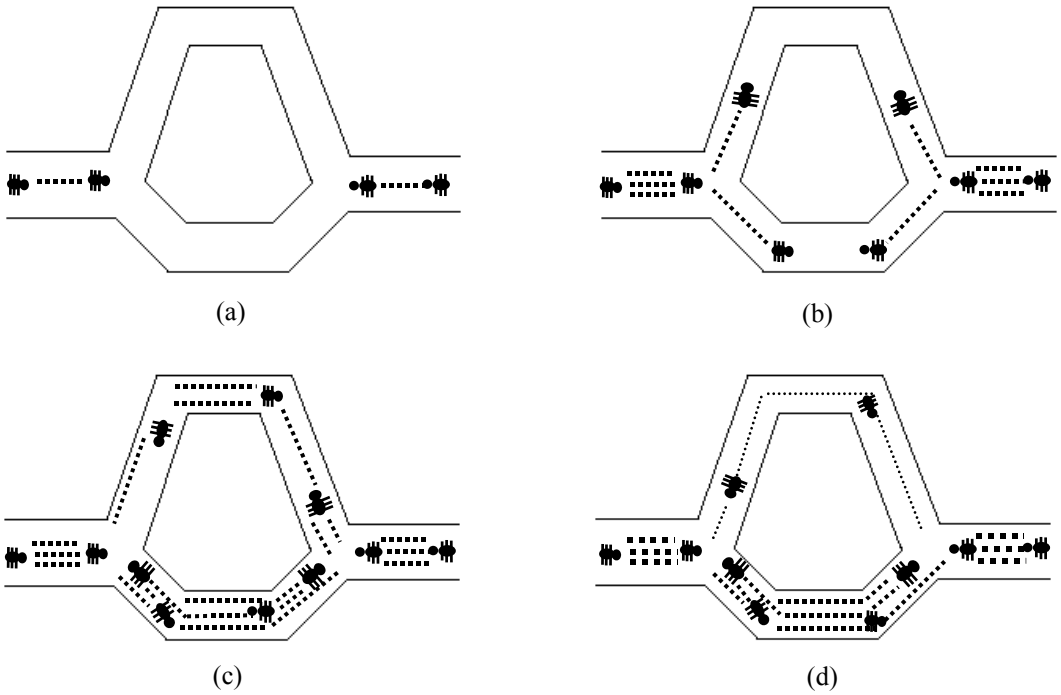

Figura 1: Comportamiento de las hormigas reales.

Basándose en el comportamiento de las hormigas arriba descrito, Dorigo et al. [1] propusieron una técnica conocida como *Ant system* para el conocido paradigma del cajero viajante (TSP – *Traveler*

Salesman Problem) [3] 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.

Trabajos posteriores de Dorigo y Gambardella extendieron los alcances del paradigma *AS*. Por ejemplo, *Ant-Q* es un híbrido entre *AS* y *Q-learning*, un conocido algoritmo de aprendizaje con realimentación positiva [4, 5].

Ant Colony System (ACS) es una extensión de *Ant-Q* [6], en el que se presentaron mejoras sustanciales a su antecesor *Ant System* en tres aspectos principales:

i) el procedimiento que realizan las hormigas para la elección de la siguiente ciudad a visitar, pues se ofrece un balance entre la exploración de nuevos caminos y la explotación del conocimiento acumulado acerca del problema;

ii) actualización global, permitiendo modificar 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 esta publicación [6] se presenta una implementación puramente secuencial, en la que se aplica además búsqueda local, utilizando el conocido método 3-opt [7] 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 siendo optimizado e intercambia las ciudades destino, evitando de este modo invertir el sentido de las ciudades visitadas.

El paralelismo es inherente al funcionamiento del algoritmo, esto es, un conjunto de agentes cooperativos intercambian información de manera indirecta, pero independientes unos de otros. En consecuencia, se vienen proponiendo varias estrategias de paralelización. Así en [8] se publican detalles de una implementación paralela síncrona y otra parcialmente asíncrona que utilizan barrera de sincronización y 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 ese momento. Se inicia una nueva iteración con el envío de la 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 este trabajo, Tomas Stützle presenta otra estrategia de paralelización [9], proponiendo corridas independientes y paralelas de un mismo algoritmo (ACO o MMAS), evitando de éste modo el *overhead* de comunicación. 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.

Por su parte, los autores del presente trabajo presentaron otra estrategia de paralelización, totalmente asíncrona [10], proponiendo que un procesador master levante a los demás procesos hijos. Cada procesador hijo realiza la computación del problema para un cierto número de hormigas, obteniendo resultados parciales que serán transmitidos a los otros procesadores siguiendo criterios de migración semejantes a los utilizados por los algoritmos genéticos paralelos, colaborando todos en la solución del problema global. Los resultados fueron obtenidos simulando procesos paralelos en una workstation DEC 3000 con procesador ALPHA, operando bajo el sistema operativo OSF/1 versión 2.0. Este trabajo fue completado en [2], presentando resultados experimentales en una red de 6 computadoras personales utilizando LINUX, en un contexto totalmente asíncrono como lo es la red de computadoras heterogéneas utilizadas en la oportunidad.

El intercambio de información entre varias colonias de hormigas ya fue presentado por Michels y Middendorf [11] para el problema de *Shortest Common Supersequence Problem (SCSP)*. En un trabajo posterior de Middendorf et al. se presentaron resultados experimentales con cuatro diferentes estrategias para el intercambio de información para el problema *TSP* [12].

Por su parte, el presente trabajo propone paralelizar el algoritmo *Ant Colony System (ACS)* en un ambiente totalmente asíncrono, en una red de computadoras personales, realizando corridas experimentales con el problema simétrico del cajero viajante (TSP) para 100 ciudades [3]. De esta manera, se tiene 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, simples agentes computacionales llamados *hormigas* trabajan en cada procesador de una red de computadoras, explorando el espacio de soluciones a la propia velocidad de cada procesador, 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, pues esto resultaría muy *costoso* en un ambiente asíncrono de computadoras heterogéneas. Con esto, cada procesador cuenta con su propia matriz de feromonas, posiblemente diferente a las matrices utilizadas en otros procesadores, que guiarán las búsquedas de buenas soluciones de forma diferenciada en cada uno de los procesadores que colaboran en la búsqueda de las soluciones globales, proporcionando una diversidad que podrá ser beneficiosa (como ocurre por ejemplo con los algoritmos genéticos) y que en general, no puede aprovecharse en las implementaciones puramente secuenciales.

La siguiente sección presenta el algoritmo *Ant Colony 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 Ant Colony System

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 agentes muy simples (llamados hormigas) que deben establecer el camino más corto para visitar todas las ciudades del problema una sola vez y regresar a la ciudad origen, para lo cual utilizan la información acumulada en una matriz de feromonas [6].

Para el presente trabajo, se considera un conjunto de n ciudades que deben ser visitadas por las m hormigas del sistema.

Para satisfacer la restricción de que una hormiga visite todas las ciudades una sola vez, se asocia a cada *hormiga k* una pequeña estructura de datos llamada lista tabú, $tabu_k$, que guarda información relativa a las ciudades ya visitadas por dicha hormiga. Una vez que todas las ciudades hayan sido recorridas, el trayecto o tour es completado volviendo a la ciudad origen. La lista tabú se vacía y nuevamente la hormiga está libre para iniciar un nuevo tour, independientemente del estado en que se encuentren las demás hormigas del sistema, lo que sugiere un alto grado de paralelismo asíncrono. En este contexto, se define como $tabu_k(n)$ al elemento n-esimo de la lista tabú de la *hormiga k* y como $J_k(i)$ al conjunto de ciudades que aun no visitó la hormiga k ubicada en la ciudad i.

Dado el conjunto de n ciudades, denominamos d_{ij} a la longitud del camino entre las ciudades *i*, *j*; para el caso del Problema de Cajero Viajante Euclidiano, tratado en el presente trabajo. 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 $\tau = \{\tau(i,j)\}$ a la matriz de feromonas a ser utilizada para consolidar la información que va siendo recogida por las hormigas. En otras palabras, $\tau(i,j)$ representa la cantidad de feromona que se va almacenando entre cada par de ciudades *(i j)*. Esta es inicializada con un valor τ_0 definida como:

$$\tau_0 = (n \times L_{nn})^{-1} \quad (1)$$

donde L_{nn} es a la longitud de un tour *típico,* obtenido inicialmente por alguna otra heurística o por pruebas aleatorias y que puede tratarse simplemente de una mala aproximación a la longitud óptima del tour, pues el algoritmo final no es muy sensible a la elección de este parámetro inicial [6].

La intensidad de las feromonas del arco *(i j)*, denotada $\tau(i, j)$, es actualizada localmente mientras las hormigas construyen su tour, esto es, al moverse de la ciudad *i* a la ciudad *j*, cada hormiga deposita una cantidad de feromonas en el arco correspondiente, calculada conforme:

$$\tau(i, j) = (1 - \rho) \times \tau(i, j) + \rho \times \tau_0 \quad (2)$$

donde $0 < \rho < 1$ es un parámetro que puede entenderse como de evaporación de las feromonas de τ (para nuestros experimentos $\rho = 0.1$). Además, se procede a una actualización global de τ según (3), la cual se realiza cuando todas las hormigas de una colonia terminaron su tour y se puede establecer la mejor solución del ciclo.

$$\tau(i, j) = (1 - \alpha) \times \tau(i, j) + \alpha \times \Delta\tau(i, j) \quad (3)$$

donde α es el coeficiente de evaporación de las feromonas, que determina el grado de influencia de una buena solución en la actualización de la matriz de feromonas, mientras que la cantidad de feromona depositada en un arco *(i j)*, está dada por:

$$\Delta\tau(i, j) = \begin{cases} (L_{gb})^{-1} & \text{si } (i,j) \in \text{al mejor tour global} \\ 0 & \text{de otra manera} \end{cases} \quad (4)$$

donde L_{gb} es la longitud del mejor tour global, hallado desde el mismo inicio de la corrida.

Durante la ejecución del algoritmo *Ant Colony System*, una *hormiga* ubicada en la ciudad i debe elegir la próxima ciudad j a visitar, para lo cual elige de entre las ciudades no visitadas, con una probabilidad q_0 la ciudad con mayor cantidad de feromonas en τ (explotación), o en forma aleatoria conforme se explica a continuación (exploración). Esto puede ser expresado como:

$$j = \begin{cases} arg\ max_{u \in J_K(i)} \left\{ [\tau(i,u)] \times [\eta(i,u)]^{\beta} \right\}, \\ \qquad \text{si } q \le q_0 \qquad \text{(explotación)} \\ R, \qquad \text{de otra manera (basado en exploración)} \end{cases} \quad (5)$$

donde q es un número aleatorio uniformemente distribuido entre [0...1] obtenido al momento de la decisión, q_0 es un parámetro $0 < q_0 < 1$ que representa una probabilidad (en nuestros experimentos se probaron varios valores para q_0 {0.1 0.5 0.7 0.9}) y R es una variable aleatoria seleccionada de acuerdo con la probabilidad dada por la ecuación (6).

$$p_k(\mathrm{i, j}) = \begin{cases} \dfrac{[\tau(i,j)] \times [\eta(i,j)]^{\beta}}{\sum_{u \in J_k(i)} [\tau(i,u)] \times [\eta(i,u)]^{\beta}} & \text{si } j \in J_k(i) \\ 0 & \text{de otra manera} \end{cases} \quad (6)$$

donde β es un parámetro que determina la importancia relativa de las feromonas respecto a la distancia $(\beta > 0)$. Por consiguiente, el algoritmo ACS secuencial puede expresarse como:

```
1. Fase de inicialización
        Inicializar contador de ciclos NC
        Para cada arco (i,j):
            valor inicial de τ_ij(t) = τ_0
        Para cada hormiga
            Elegir ciudad origen

2. Repetir hasta llenar tabu_k
      Si es la ultima posición de tabu_k
          Para cada hormiga:
              Insertar ciudad origen en tabu_k
      Sino
          Para cada hormiga:
              Elegir próxima ciudad a ser visitada según ecuaciones (5) y (6)
              Mover la hormiga a la próxima ciudad
              Insertar ciudad seleccionada en tabu_k
      Para cada hormiga:
          Actualizar feromonas según ecuación (2)

3. Repetir para cada hormiga k
      Calcular la longitud L_k del ciclo
      Actualizar feromonas según ecuación (3)

 4. Si (Condición_fin = Verdadero)
      Imprimir camino más corto L_k
   Sino
      Ir a la Fase 2
```

Pseudocódigo 1: Algoritmo *Ant Colony System* secuencial.

En resumen, ACS se inicia ubicando *m* hormigas en *n* ciudades de acuerdo a una regla de inicialización. Para nuestros experimentos se eligen las ciudades origen de manera aleatoria. Cada hormiga construye su propio tour, eligiendo la próxima ciudad a visitar aplicando las ecuaciones (5) y (6). Mientras construyen su tour, las hormigas actualizan feromonas al moverse de la ciudad *i* a la ciudad *j* según la ecuación (2). Cuando todas las hormigas volvieron a su ciudad origen, se calcula la mejor distancia hallada en el ciclo, y nuevamente se realiza una actualización de las feromonas, esta vez teniendo en cuenta, solo la mejor solución global L_{gb} según la ecuación (3).

El proceso se repite iterativamente hasta que se cumpla algún criterio de parada. Por simplicidad, en este trabajo, el proceso termina si el contador de ciclos alcanza un número máximo de ciclos *NCMAX* (definido por el usuario), para nuestros experimentos igual a 10.000.

Debido a la forma en que se calculan las probabilidades en (5) y (6), las hormigas son guiadas en la búsqueda de soluciones por información heurística, prefiriendo elegir los arcos con menor distancia y mayor cantidad de feromonas, lo que a su vez sirve para incrementar la cantidad de feromonas en los caminos que van resultando óptimos a medida que avanza la corrida del algoritmo.

3 Implementación paralela

Ant Colony 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 en realidad hace su trabajo en forma independiente, tomando sus propias decisiones sobre la base de su ambiente localizado (ej. feromona percibida). Cada individuo comunica indirectamente a las demás hormigas su experiencia a través de la sustancia química denomina feromonas, por lo que existe un paralelismo natural en el comportamiento de las hormigas. Análogamente, el presente trabajo propone la utilización de agentes computacionales asíncronos y paralelos, llamados *hormigas*. Cada uno de estos

agentes va construyendo su tour en el computador en el que se encuentra, 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. Como en general, el número de hormigas que conforman una colonia es bastante superior al número de computadoras personales disponibles, el presente trabajo propone que cada procesador realice la computación del problema para cierto número de hormigas, de la misma forma que se lleva a cabo en un contexto secuencial, obteniendo resultados parciales que al cabo de cierto número de ciclos serán trasmitidos a los otros procesadores (sin ninguna sincronización), con el fin de colaborar todos en la solución del problema global, eligiendo para esto como hormigas migrantes a aquellas que hayan obtenido las mejores soluciones en su procesador.

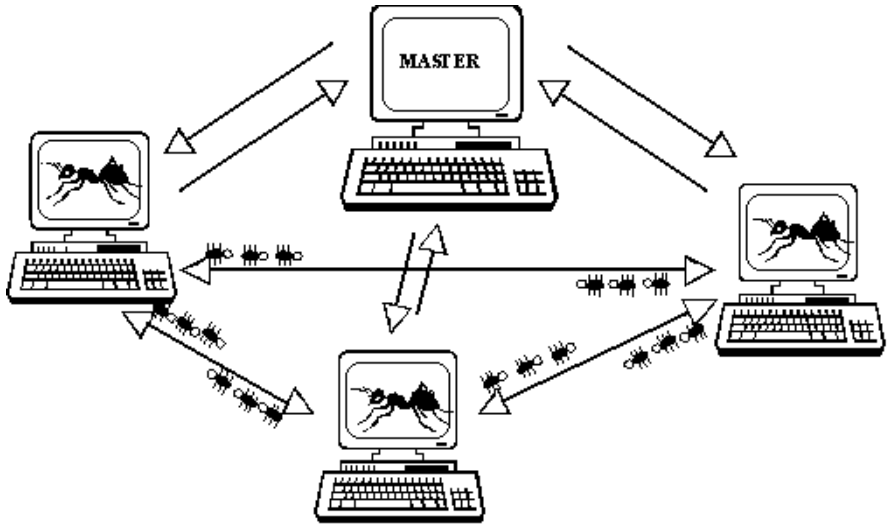

Figura 2: Estrategia de paralelización.

El asincronismo aquí propuesto elimina los tiempos muertos producidos por la espera en la sincronización de la comunicación debido al importante cuello de botella que representa la sincronización en las implementaciones paralelas del Pseudocódigo 1, en especial, debido a las necesidades de sincronización de la Fase 3. En consecuencia, el presente trabajo propone que las hormigas migren de un procesador a otro en forma asíncrona, respetando políticas migratorias similares a las utilizadas con los Algoritmos Genéticos Paralelos, políticas definidas por los siguientes parámetros [13]:

- *El intervalo de migración*: que establece cada cuantos ciclos cierta cantidad de hormigas migrarán de un procesador a otro. Para nuestros experimentos es un parámetro fijado en 8 ciclos, aunque puede tomar otros valores, que no son presentados en este trabajo.

- *La tasa de migración*: que indica cuantas hormigas han de comunicarse a los otros procesadores destino cuando se cumpla el intervalo de migración. En las experiencias aquí reportadas, solo migra la mejor solución.

- *El criterio de selección*: que determina la política a seguir para la selección de las hormigas que han de migrar. Por ejemplo que sean elegidas al azar o las mejores, siendo este último el implementando en las experiencias que se reportan a continuación.

Cuando cada proceso recibe a las hormigas migrantes, de otros procesadores, estas compiten con la colonia local de hormigas a fin de que solamente la mejor solución de todas actualice la matriz de feromonas.

En consecuencia, esta propuesta realiza corridas independientes del algoritmo en cada procesador hijo, que utiliza su propia matriz de feromonas, lo que resulta en procesadores independientes (no sincronizados) con diferentes características de búsqueda, dado que en general las matrices de feromonas son diferentes en cada procesador, pero que trabajan en equipo al compartir buenas soluciones que llevan en general a la convergencia de las matrices de feromonas a una misma solución global de longitud L_{gb}, dado que a la larga, todas ellas van actualizando τ utilizando el mismo tour de longitud L_{gb} al aplicar la ecuación (3).

Para levantar y controlar todos los procesos hijos donde realmente se hace el trabajo de optimización, se propone un master que comunique los parámetros iniciales y el número de ciclos que deben realizar cada uno de los procesos hijos, y espera de estos la comunicación de que han atendido al criterio de finalización (ver Figura 2) para dar por terminada la corrida.

1. Inicio
   Levantar procesos
   Enviar parámetros a cada hijo
   Fin= Falso
2. Repetir mientras no sea Fin
   Recibir solución de los hijos
   Guardar mejor solución
   Si suficientes procesos terminaron
   Fin = Verdadero
3. Eliminar procesos hijos

Pseudocódigo 2: Proceso Master.

1. Inicio
   Recibir parámetros
   Inicializar variables
2. Repetir mientras (Ciclo_actual < *NCMAX*)
   Mover hormigas hasta completar tour
   Calcular distancia recorrida para cada hormiga
   Escoger hormigas a emigrar
   Enviar hormiga migrante a otros procesos
   Recibir hormigas migrantes de otros procesos
   Seleccionar la hormiga que actualizará feromonas
   Actualizar matriz de feromonas
   Guardar camino más corto
   Ciclo_actual = Ciclo_actual+1
3. Enviar mejor solución al master.

Pseudocódigo 3: Proceso Hijo.

El Pseudocódigo 2 presenta al Master de la versión paralela que se encarga de administrar la implementación, incluyendo el lanzamiento de los procesos Hijos, presentado en el Pseudocódigo 3. Como la plataforma utilizada es homogénea, el Master eliminaba los procesos hijos, solo cuando todos comunicaron que ya realizaron NCMAX ciclos. Sin embargo, en un contexto heterogéneo, se deberá definir que significa "suficiente" (ver paso 2 del Pseudocódigo 2); por ejemplo, parando la corrida cuando K de N procesos avisan al Master que ya realizaron los ciclos que le fueron asignados.

4 Resultados experimentales

Los experimentos fueron realizados en una red local tipo *Ethernet* (10 Mbps) con cuatro computadoras personales homogéneas, con procesadores AMD-K6 de 350 MHz, 128 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 tipo utilizado para probar las distintas variantes del ACS fue el KroA100, *TSP* para 100 ciudades, cuyos datos se pueden encontrar en [3].

Se realizaron corridas de NCMAX=10.000 ciclos cada una, tanto para las implementaciones secuenciales como para las implementaciones paralelas (con cuatro procesadores), en donde cada procesador corre un número de ciclos igual a NCMAX / (Nro. de procesadores). Para los resultados que siguen $\rho = \alpha = 0.1$ por simplicidad. El número de ciclos por procesador puede variar en un ambiente heterogéneo, para balancear la carga de los procesadores, es decir, se puede utilizar un número de ciclos proporcional a la velocidad de procesamiento. De hecho, los autores ya han realizado experiencias con *Ant System* en redes heterogéneas con 12 computadoras personales no homogéneas, ubicadas en 2 redes locales interconectadas, logrando el balance de carga mediante el uso de un número de ciclos proporcional al desempeño de cada procesador [14].

A continuación, se presentan resultados experimentales para las siguientes versiones:

- ACS1 que utiliza el conocido método de búsqueda local 3-opt [7], aplicado solo cuando se halla una solución mejor que la almacenada como óptima de la corrida, realizando tres cortes en el tour hallado e intercambiando las ciudades destino con el fin de encontrar mejores soluciones.

- ACS2 que también aplica búsqueda local 3-opt pero eligiendo los arcos a ser cortados de manera aleatoria, solo un número *K* de veces, de forma a invertir menos tiempo relativo en la búsqueda heurística.

- ACS3 que implementa el algoritmo original de Dorigo et al. sin búsqueda local, de forma a verificar el grado de optimización que se logra cuando solo se utiliza la colonia de hormigas;

- ACS3' que modifica ligeramente el ACS3 implementando una reinicialización de la matriz de feromonas después de 100 ciclos sin que se haya hallado una nueva mejor solución, pero recordando la mejor solución global, con la esperanza de incrementar la exploración una vez que la matriz de feromonas produzca un mayor nivel de explotación por acercarse a la convergencia.

- ACS4 que es aplicable básicamente a un contexto paralelo (por lo que no se lo implementó en forma secuencial), pues cada procesador hijo corre un algoritmo diferente [13], esto es, el procesador 1 utiliza el ACS1, el procesador 2 utiliza el ACS2, el procesador 3 utiliza el ACS3 y el procesador 4 utiliza el ACS3'.

Versión $q_0 = 0.9$	**Secuencial**			**Paralelo**		
	Tiempo	Distancia	Mejor Sol	Tiempo	Distancia	Mejor Sol
ACS1	2112.5	22273.9	21737	564.6	21895.10	21331
ACS2	2274.2	22077.6	21791	547.8	21947.60	21529
ACS3	2157.7	22159.8	21718	551.2	22106.90	21558
ACS3'	2163.8	21717.7	21467	571.5	21848.30	21510
ACS4	-----	-----	-----	546.1	21874.80	21356
Promedio	*2177.1*	*22057.3*	*21678.3*	*556.2*	*21934.5*	*21456.8*

Tabla 1: Promedios de Tiempos y Distancias en 10 corridas típicas.

En la Tabla 1 se pueden observar los promedios en tiempo y distancia hallados para cada una de las diferentes versiones en 10 corridas, así como la mejor solución encontrada en el experimento, todas utilizando el parámetro $q_0 = 0.9$. La última fila ilustra el promedio obtenido entre todas las versiones (4 para el caso secuencial y 5 para el paralelo) de forma a comparar cada uno de los parámetros de desempeño. Como pude notarse, las versiones paralelas logran en general mejores soluciones que las versiones secuenciales (ver promedios de distancias medias o de mejores soluciones), con una aceleración superior a 3.9, lo que resulta en una eficiencia del orden del 98 % en el uso de los procesadores.

Versión	**Secuencial**			**Paralelo**		
	Tiempo	Distancia	Mejor Sol	Tiempo	Distancia	Mejor Sol
ACS1	2581.4	21636	21380	662.3	21692	21417
ACS2	2620.2	21802.8	21556	670.4	21725	21390
ACS3	2611.4	21749.9	21477	667.7	21650	21383
ACS3'	2676.1	21747.6	21451	668.0	21740	21387
ACS4	-----	-----	-----	637.1	21617	21331
Promedio	*2622.3*	*21734.1*	*21466.0*	*661.1*	*21684.8*	*21381.6*

Tabla 2: Promedios de Tiempos y Distancias.

Por su parte, la Tabla 2 muestra los promedios obtenidos para cada experimento tanto secuencial como paralelo utilizando como parámetro $q_0 = 0.1$ permitiendo al algoritmo una mayor búsqueda exploratoria. Sólo en la versión ACS4 se permitió a cada algoritmo utilizar un valor diferente para el parámetro q_0. Así tenemos que, el procesador 1 corriendo ACS1 utilizó $q_0 = 0.1$, el procesador 2 corriendo ACS2 con $q_0 = 0.5$, el procesador 3 corriendo ACS3 con $q_0 = 0.7$ y el procesador 4 con ACS3' y $q_0 = 0.9$. Nuevamente se nota que las versiones paralelas son en promedio mejores que las secuenciales, dado que obtienen soluciones al menos tan buenas, pero con aceleraciones promedio que en este caso llegan a casi 3.97, lo que resulta en una eficiencia promedio que supera los 99 %.

El excelente nivel de eficiencia que se ha obtenido (mayor a 99%) no es de extrañar, pues como ya fuera expuesto, el ACS es inherentemente paralelo y al implementarse en forma asíncrona se elimina los retardos de sincronización, único obstáculo práctico para lograr niveles excelentes de paralelismo al considerar que ningún procesador requiere de resultados de otros procesadores para realizar sus cálculos, lo que incluso nos permite postular que el sistema es además tolerante a fallas, dado que la caída de un proceso hijo no impide que se obtenga la solución deseada.

Sin embargo, al considerar la escalabilidad del sistema propuesto a un importante número de procesadores, el costo de la comunicación de un resultado a todos los procesadores *(broadcasting)* puede llegar a ser prohibitivo por lo que se deberán definir políticas migratorias en base a lo presentado en la sección 3, lo que puede implicar una pérdida de eficiencia respecto a los excelentes resultados aquí reportados.

Finalmente, cabe mencionar la aparente ventaja de poder utilizar versiones (y hasta algoritmos) diferentes en cada procesador, pues con ello el conjunto resulta en principio al menos tan bueno como el mejor de los algoritmos secuenciales siendo combinados, aunque en nuestros experimentos, los datos no resultaron lo suficientemente concluyentes como para afirmar que en efecto esta mejora es considerable en la práctica.

5 Conclusión

El presente trabajo propone una implementación paralela asíncrona del ACS (*Ant Colony System*), originalmente propuesto por Dorigo y Gambardella [6, 15] en un contexto puramente secuencial. Sin entrar a analizar las bondades del ACS frente a otras alternativas computacionales ampliamente difundidas en la literatura, la propuesta está especialmente diseñada para lograr un alto grado de eficiencia en las redes de computadoras personales, normalmente disponibles en la mayoría de las instituciones. Para esto, cada computadora procesa una población local, a su propio ritmo (y posiblemente, con su propio algoritmo), sin ningún tipo de sincronización con los demás procesadores, enviando buenas soluciones a otros procesadores y recibiendo de estos sus mejores soluciones (sin bloqueos de sincronización), de forma a lograr una sinergia entre todos los procesadores que colaboran en la búsqueda de la solución óptima.

En efecto, el trabajo presenta resultados experimentales aplicados al problema del cajero viajante que comprueban el alto grado de eficiencia de la implementación propuesta, llegando en algunos casos (ver Tabla 2) a superar el 99% (una aceleración promedio de casi 3.97 con 4 procesadores). Al mismo tiempo, la implementación propuesta permite utilizar versiones (y hasta algoritmos) diferentes en cada procesador, lo que en principio trae ventajas adicionales al permitir una mayor exploración, pues aun con la misma matriz de feromonas inicial, cada procesador podrá realizar su búsqueda en otra región. Sin embargo, se requiere todavía de más experimentos para confirmar las ventajas de utilizar diferentes versiones en cada procesador.

Si bien por un tema de espacio el presente trabajo solo presentó resultados experimentales utilizando 4 procesadores, se destaca que ya se han realizado experiencias con hasta 12 procesadores distribuidos en dos redes locales interconectadas por un *router* [12], con básicamente los mismos resultados, es decir, un excelente nivel de paralelismo sin perjudicar la calidad de las soluciones obtenidas. En consecuencia, se puede postular la escalabilidad de la implementación propuesta, utilizando redes con un gran número de computadoras personales, toda vez que el tamaño del problema así lo justifique.

Finalmente, cabe mencionar que el paralelismo implícito en una Colonia de Hormigas permite múltiples aplicaciones que pueden llegar a beneficiarse de esta importante característica, especialmente cuando aprovechado en el contexto de grandes redes de computadoras [16]. En consecuencia, los autores se encuentran estudiando otras aplicaciones del ACS asíncrono aquí presentado, como por ejemplo, las aplicadas al ruteo óptimo de vehículos. Además, se encuentran realizando experimentos para evaluar el desempeño de distintas versiones en un ambiente heterogéneo así como distintas instancias del problema del cajero viajante [3].

Bibliografía

[1] 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, pg. 1-13, 1996.

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

[3] Reinelt G., "TSP", http://www.iwr.uniheidelberg.de/groups/comopt/software/TSPLIB95/ TSPLIB.html. Universität Heidelberg, Institut für Angewandte Mathematik, Alemania. 2000.

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

[5] 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, pg. 656-665, 1996.

[6] 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, pg. 53-66, 1997.

[7] 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.

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

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

[10] Almirón M., Chaparro E. y Barán B. "Sistema distribuido de hormigas para el problema del cajero viajante" *XXV Conf. Latinoam. de Informática* – CLEI'99, Paraguay, 1999.

[11] Michels R y Middendorf M. "An ACO Algorithm for the Shortest Common Supersequence Problem" *Fifth International Conference on Paralell Problem Solving from Nature (PPSN'98)*, Amsterdam, Springer-Verlag, LNCS 1498, 692-701, 1998.

[12] Middendorf M, Reiscgke F.y Schmeck H. "Information Exchange in Multi Colony Ant Algorithms" Parallel and Distributed Computing, *Proceedings of the 15 IPDPS 2000 Workshops, Third Workshop on Biologically Inspired Solutions to Parallel Problems (BIOSP3)*, Cancum, Mexico, Springer-Verlag LNCS 1800, 645-652, 2000.

[13] 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.

[14] Almirón M. y Barán B. Reporte Técnico RT003-2001. Universidad Nacional de Asunción, Paraguay, 2001. URL: http://www.cnc.una.py/invest/ publicaciones.html.

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

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