Paralelización de un Algoritmo de Búsqueda Local Iterada para el Problema del Agente Viajero

Rodrigo Eugenio Morales-Navarro[1], Marco Antonio Cruz-Chávez[2], Rafael Rivera-López[3], Abelardo Rodríguez-León[3], Alina Martínez-Oropeza[2] y Pedro Moreno Bernal[2]

[1,3]Departamento de Sistemas y Computación. Instituto Tecnológico de Veracruz
Calzada Miguel Ángel de Quevedo 2779, Veracruz, México
[2]CIICAp, Universidad Autónoma del Estado de Morelos
Av. Universidad 1001, Col. Chamilpa, Cuernavaca, Morelos, México
[1]monroe.eskinka@gmail.com [2]{mcruz, alinam}@uaem.mx [3]{arleon,rrivera}@itver.edu.mx

Resumen. Este trabajo presenta un enfoque de paralelización de un algoritmo de búsqueda local iterada que utiliza una estructura de vecindad híbrida para construir rutas del problema del agente viajero. Este enfoque se implementa sobre un par de clusters ubicados en dos ciudades de la República Mexicana (Cuernavaca y Veracruz). El algoritmo utiliza dos niveles de paralelización que hace uso de threads en cada nodo de un cluster, así como de un mecanismo de paso de mensajes para la paralelización a nivel de cluster. Los resultados obtenidos al aplicar este enfoque muestran que, haciendo uso de clusters, se produce una reducción en el tiempo de ejecución, así como se obtiene una sustancial mejora en la solución encontrada, ya que se realiza una mayor explotación del espacio de soluciones del problema.

1 Introducción

En México, desde hace algunos años varias instituciones de investigación y de educación superior han invertido en la construcción de clusters de computadoras que se aplican en el desarrollo de proyectos de investigación en varias áreas de conocimiento ([1] y [2] por ejemplo). Derivado de una iniciativa para crear un laboratorio nacional de grids por las principales instituciones de educación superior en México, el Instituto Tecnológico de Veracruz (ITVer) y la Universidad Autónoma del Estado de Morelos (UAEM) unieron esfuerzos para compartir los recursos de sus clusters de alto rendimiento logrando construir una grid de cómputo [3].

La optimización combinatoria es un área que agrupa muchos problemas computacionalmente complejos y que tienen como objetivo encontrar el óptimo de una determinada función sobre un conjunto finito de soluciones donde las variables han de ser discretas, restringiendo su dominio a una serie finita de valores. Habitualmente, el número de soluciones en estos problemas es muy elevado, haciendo inviable la evaluación de todas ellas para encontrar el óptimo [4]. El uso de la paralelización para resolver problemas de optimización combinatoria es un enfoque que ha ganado importancia en años recientes [5]. Por ejemplo, en [6] se propone un algoritmo evolutivo paralelo que usa recocido simulado para resolver el Job Shop Scheduling Problem

M.A. Cruz-Chávez (Ed): CICos 2011, ISBN. 978-607-00-5091-6. pp. 16 – 25, 2011

(JSSP) y en [7] se describe un algoritmo paralelo de recocido simulado para resolver el problema de máquinas paralelas no relacionadas ponderadas.

Dentro de los problemas clásicos de optimización combinatoria se encuentra el Problema del Agente Viajero (TSP por sus siglas en inglés) que, por su complejidad y naturaleza combinatoria, para su resolución generalmente se emplean heurísticas que implementan estructuras de búsqueda local [8], las cuales han resultado ser eficientes en la búsqueda de buenas soluciones. Dentro del proceso de búsqueda de soluciones, varias técnicas de construcción de rutas del TSP se han propuesto a través del uso de diferentes estructuras de vecindad y mecanismos de búsqueda local ([9], [10], [11] y [12]). En [12] se presenta una propuesta de vecindad híbrida que combina diferentes estructuras de vecindad para encontrar mejores soluciones dentro de un algoritmo de búsqueda local iterada.

Este trabajo presenta un enfoque de paralelización del algoritmo de búsqueda local iterada presentado en [12] que evalúa de manera concurrente nuevas rutas. Este enfoque se implementa sobre dos clusters de computadoras. El algoritmo implementa dos niveles de paralelización que hace uso de threads en cada nodo de un cluster y de un mecanismo de paso de mensajes para la paralelización en un cluster. Los resultados obtenidos al aplicar este enfoque muestran que, haciendo uso de clusters, se produce una reducción en el tiempo de ejecución, así como se obtiene una sustancial mejora en la solución encontrada, ya que se realiza una mejor explotación del espacio de soluciones del problema.

El resto de este trabajo se organiza como sigue: En la siguiente sección se describen los conceptos de búsqueda local para TSP, para luego presentar la descripción de la propuesta de paralelización, después se hace una descripción de los clusters utilizados en las pruebas y finalmente se analizan los resultados de este enfoque.

2 Búsqueda local para el problema del agente viajero

El problema del agente viajero consiste en encontrar la ruta más corta que un agente debe seguir para visitar un grupo de ciudades que, comenzando y terminando en una ciudad concreta, pase una sola vez por cada una de las ciudades [13]. En [14] se mencionan algunas de las aplicaciones planteadas de manera análoga al TSP, como por ejemplo la programación de tareas en una máquina, la logística en la repartición de mercancía a los clientes y la revisión de medidores en las casas de una colonia. TSP se representa como un grafo donde las ciudades son los nodos y las distancias entre las ciudades se representan en los arcos del grafo (figura 1). Como ya se indico, en la actualidad existen algoritmos que resuelven instancias de TSP haciendo uso de metaheurísticas que realizan una búsqueda por vecindad dentro del espacio de búsqueda de soluciones, permitiendo encontrar soluciones muy cercanas a la óptima pero en menor tiempo.

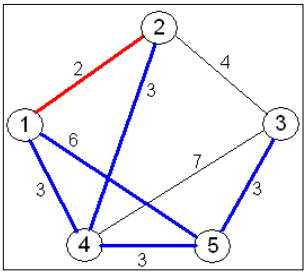

Fig. 1.- Grafo del problema del agente viajero.

Los algoritmos de búsqueda por vecindad (también llamados algoritmos de búsqueda local) constituyen un tipo bastante amplio de algoritmos de mejora en los que en cada iteración se obtiene una solución mejorada buscando en la "vecindad" de la solución existente de un problema de optimización combinatoria. En un procedimiento de búsqueda local se entiende por vecindad a un conjunto de soluciones cercanas a una solución inicial dada (figura 2). Matemáticamente se expresa como que dado un punto factible sÎS en una instancia del problema, la vecindad de s se define como el conjunto N(s) de puntos factibles y cercanos a s; dicho conjunto indica que cada solución s'ÎN(s) puede ser encontrada directamente desde s en un solo paso [10].

La selección de una estructura de vecindad apropiada es un aspecto crítico para el diseño de esta clase de algoritmos que permite una mejor exploración y explotación del espacio de soluciones. El algoritmo de búsqueda local iterada presentado en [12] cambia en cada iteración el criterio de vecindad determinando aleatoriamente el uso de una vecindad de posibles rutas de TSP entre los criterios de par adyacente, par aleatorio, dos pares adyacentes y dos pares aleatorios (figura 3). La figura 4 muestra la estructura del algoritmo de búsqueda local iterada con criterio de vecindad híbrida.

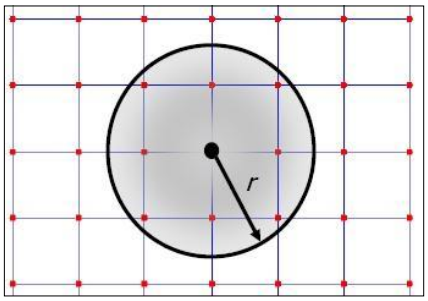

Fig. 2.- Vecindad de un punto con distancia unitaria.

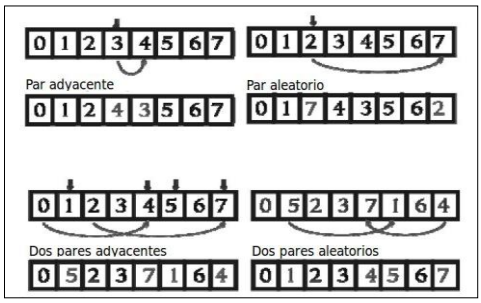

Fig. 3.- Criterios de vecindad usados en [12].

En la figura 4, los métodos `leer_archivo()`, `crea_matriz()`, `crea_vector()` y `libera()` son métodos para construir el problema y aplicar el algoritmo. El método `permuta()` es el que se encarga de seleccionar una ruta vecina a evaluar utilizando alguno de los criterios mostrados en la figura 3, el método `distancias()` calcula la distancia de la ruta seleccionada de la vecindad y el méto-

do `compara()` actualiza la solución actual cuando la ruta vecina tiene una menor distancia a la actual.

```
main(){
     leer_archivo();
     crea_matriz();
     crea_vector();
     do{
       permuta();
       distancias();
       compara();
     } while (criterio_de_paro);
     libera();
}
```

Fig. 4.- Estructura del algoritmo de búsqueda local iterada con criterio de vecindad híbrida.

3 Propuesta de paralelización con threads y paso de mensajes

Para obtener una versión paralela del algoritmo de búsqueda local iterada con criterio de vecindad híbrida se realizó un análisis del tiempo de ejecución (profile). A partir de los resultados mostrados en la figura 5, se determinó que la función a paralelizar es la que realiza el cálculo de las distancias debido a que es la que mayor tiempo computacional consume de manera individual. La ejecución del algoritmo secuencial original aplicaba una búsqueda limitada a 5 minutos, realizando en ese tiempo alrededor de 834,000 iteraciones (como se muestra en la figura 5) y produciendo una menor distancia de 194,498 para un grupo de 4,000 ciudades.

La propuesta de paralelización de este trabajo se estructura en dos niveles: usando memoria compartida y usando memoria distribuida.

% time	cumulative seconds	self seconds	calls	self ms/call	total ms/call	name
99.88	291.33	291.33	834137	0.35	0.35	distancias
0.07	291.54	0.21	1	210.00	210.00	leer_archivo
0.04	291.65	0.11	834137	0.00	0.00	permuta
0.01	291.67	0.02				main
0.00	291.67	0.00	834137	0.00	0.00	compara
0.00	291.67	0.00	1	0.00	0.00	crea_matriz
0.00	291.67	0.00	1	0.00	0.00	crea_vector
0.00	291.67	0.00	1	0.00	0.00	libera

Fig. 5.- Perfil de la versión secuencial del algoritmo de búsqueda local iterada.

a) Memoria Compartida: Como el cálculo de la distancia de la ruta requiere el uso de "una matriz de distancias" que contiene las distancias entre cualquier par de ciudades del problema, si se divide el cálculo de la ruta en varios procesos concurrentes, estos procesos utilizarán simultáneamente una única matriz de distancias, por lo que el enfoque debería ser de memoria compartida. A partir del profile, se efectuó una división del trabajo del cálculo de la distancia de la ruta en más de un hilo, para lo cual se

empleó OpenMP. En la figura 6 se muestra un ejemplo del uso de 4 threads para calcular la distancia de una ruta generada para 4,000 ciudades.

b) Memoria distribuida: El enfoque propuesto en [12] iniciaba con una ruta aleatoria para iniciar la búsqueda. Para aprovechar el poder de cómputo de un cluster de computadoras, cada nodo puede realizar la búsqueda a partir de una ruta inicial; entonces, en el enfoque paralelo, se pueden inicializar tantas rutas aleatorias como nodos existan en el cluster y tomar la mejor al final el proceso de búsqueda local en cada nodo del cluster. Con este enfoque lo que se produce es una mejora en la explotación del espacio de búsqueda, ya que se realizan búsquedas concurrentes con rutas iniciales diferentes. La figura 7 presenta un ejemplo donde se generan cuatro caminos aleatorios y se envían a cuatro nodos. En este enfoque se hace necesario definir un nodo como maestro, que es el que generará las rutas iniciales y determinará cual ruta producida por los otros nodos es la mejor.

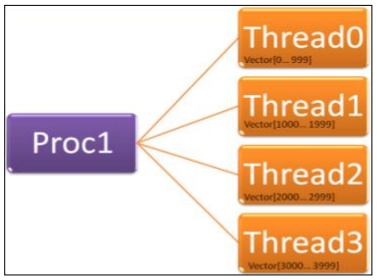

Figura 6.- Paralelización por memoria compartida.

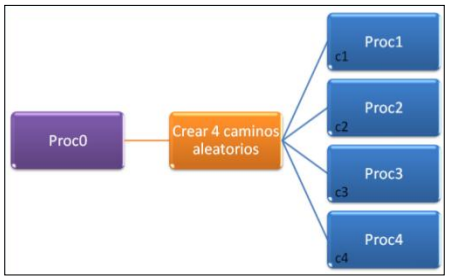

Figura 7.- Paralelización por paso de mensajes.

Además, a la propuesta se le añadió que al momento de que el nodo maestro obtenga las rutas resultantes, realice una mezcla de las soluciones y envié las rutas generadas nuevamente a los nodos, los cuales de nuevo realizarán la búsqueda de una menor ruta y devuelvan al proceso maestro la mejor solución encontrada. Este proceso se deberá realizar un número determinado de iteraciones. La figura 8 presenta el grafo de actividades del enfoque de paralelización propuesto.

En la figura 8, la actividad T.1.1 se realiza en todos los nodos para inicializar las variables, vectores y matrices necesarias para realizar las operaciones requeridas durante las iteraciones. Además, se carga en memoria la matriz de distancias entre las ciudades, la cual se encuentra en un archivo de texto.

El nodo P0 (nodo maestro) es el único que realiza la actividad T.1.2, ya que es el que se encarga de centralizar todas las rutas para determinar la de menor distancia en cada iteración, así como realizar la mezcla de rutas. En la primera iteración el nodo 0 crea N rutas aleatorias que serán enviadas a los demás nodos (T.1.3) y de esta manera cada nodo itera con una ruta diferente. Una vez que se han enviado las rutas, el nodo 0 se queda en espera de recibir de vuelta la ruta de cada uno de los otros nodos.

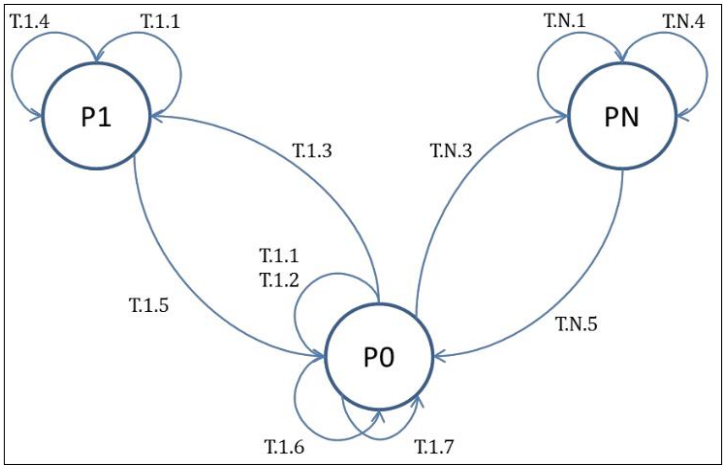

Figura 8.- Grafo de estados del enfoque de paralelización de dos niveles.

La actividad T.1.4 es realizada por todos los nodos, excepto por el nodo maestro, y es en este punto en donde se lleva a cabo el trabajo más pesado del algoritmo. Cada nodo realiza una serie de permutaciones a la ruta que se le asignó, calculando por cada cambio la distancia de la nueva ruta generada y comparándola con la anterior para que al final del ciclo envíen al nodo 0 la ruta con la menor distancia que se haya generado (T.1.5).

Una vez que el nodo 0 recibe todos las rutas de vuelta, los almacena en una matriz y las distancias se almacenan en un vector, en la posición correspondiente con la ruta (T.1.6). Posteriormente determina la menor distancia y la almacena, así como su respectiva ruta. El nodo 0 realiza la mezcla de las rutas considerando que deben generarse siempre rutas válidas para la siguiente iteración.

Si aún no se ha ejecutado el número de iteraciones establecidas, se repite el ciclo desde la actividad T.1.3, con la diferencia que se envían los caminos resultantes de la mezcla. Después de que se hayan realizado el número de iteraciones, el nodo 0 imprime el vector con la ruta de menor distancia encontrado en toda la ejecución y su costo. Finalmente, todos los nodos liberan la memoria ocupada.

4 Clusters utilizados para implementar en enfoque

En el año 2008, con la colaboración de los cuerpos académicos de Optimización y Software de la UAEM (UAEMOR-CA-87) y de Cómputo Intensivo Aplicado a la Ingeniería (ITVER-CA-1) se implementa una grid, denominada Tarántula, entre las dos instituciones académicas. Los clusters que se utilizan en este trabajo, y que pertenecen a la grid Tarántula son el cluster Nopal del ITVer (figura 9) y el cluster CIICAp de la UAEM (figura 10). Las características de estos clusters se presentan en la tabla 1.

Fig. 9.- El cluster Nopal del ITVer.

Fig. 10.- El cluster CIICAp de la UAEM.

Tabla 1.- Características de los clusters de la grid Tarántula utilizados en este trabajo.

Cluster	Descripción
Nopal	Tiene 15 computadoras (dual y quad-core), el front-end tiene un velocidad de procesamiento de 3.2 GHz y 14 nodos con una velocidad de 2.33 GHz La capacidad de almacenamiento del cluster es de 1.2 TB y una memoria total de 57 GB. El sistema operativo es Rocks 5.2.
CIICAp	Este cluster cuenta con un front-end con procesador Pentium 4 a 2.8 GHz, y 18 nodos esclavos procesador Intel Celeron Dual Core a 2.0 GHz La capacidad de almacenamiento del cluster es de 2.9 TB y una memoria total de 36 GB. También usa el sistema operativo Rocks 5.2

5 Análisis de Resultados

Para el código paralelo se estableció la evaluación de 60 rutas durante 3 iteraciones haciendo uso de 10, 20, 30 y 40 procesos para sacar una estadística y analizar el comportamiento del algoritmo en diferentes ámbitos. Además, haciendo la comparación con el tiempo de ejecución del código secuencial, se obtuvo el speed up para la ejecución en 10, 20 y 30 procesos haciendo uso de nodos dual core, nodos quad core y de una combinación de ambos.

En la figura 11 se puede apreciar la ganancia en tiempo de la ejecución paralela frente a la ejecución secuencial. Como es de esperarse, la generación de 60 rutas haciendo uso de 30 procesos arroja un resultado mucho menor que el secuencial (se redujo el tiempo de ejecución aproximadamente 23 veces comparado con la ejecución secuencial usando nodos quad core).

Por otro lado, en la figura 12 se muestra el speed up obtenido en las distintas pruebas de la ejecución paralela. Se puede apreciar que el speed up varía dependiendo de la combinación de los nodos de los clusters de Tarántula. En estas pruebas se puede observar que haciendo uso de una mayor cantidad de nodos dual core se obtiene un mejor speed up que utilizando solamente nodos quad core; esto se debe a que los pri-

meros tienen un velocidad de 2.33 GHz, mientras que los nodos dual core tienen una velocidad de 2.66 GHz

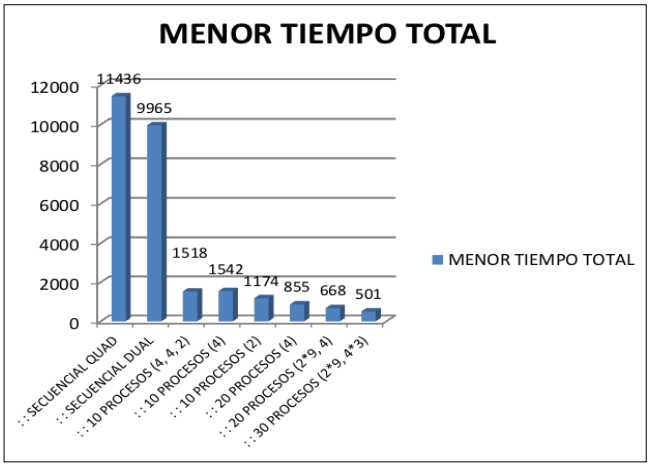

Fig. 11.- Tiempo de ejecución de la versión paralela del algoritmo.

Además, se aprecia que la curva del incremento del speed up resulta ser como se esperaba, obteniéndose una eficiencia del 66% haciendo uso de 30 procesos (9 nodos dual core y 3 quad core).

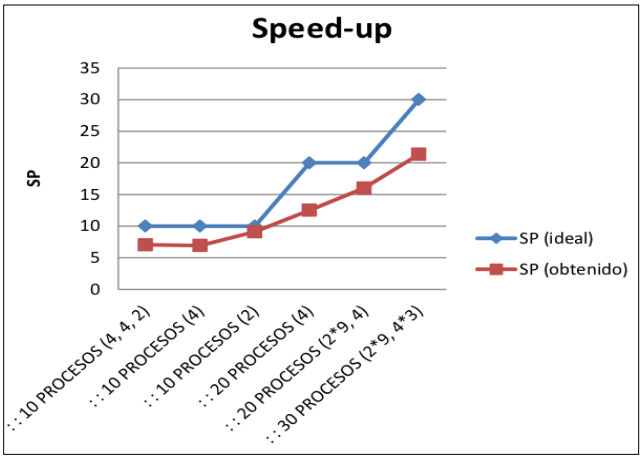

Fig. 12.- SpeedUp de la versión paralela del algoritmo.

Finalmente, en la figura 13 se muestra la gráfica con la distancia menor encontrada en las distintas pruebas realizadas. La menor distancia encontrada en las pruebas con el algoritmo secuencial fue de 187,792, mientras que en las pruebas del código paralelizado fue de 186,025. Cabe aclarar que las pruebas se realizaron sobre una instancia del problema del agente viajero de 4,000 ciudades generadas aleatoriamente.

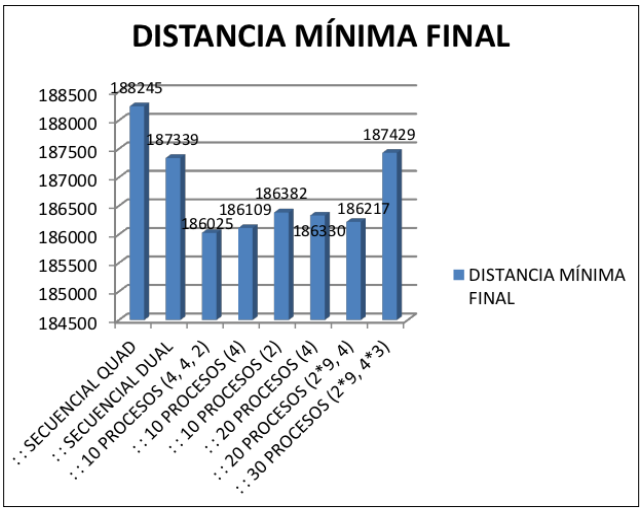

Fig. 13.- Distancias mínimas encontradas por la versión paralela del algoritmo.

Conclusiones y trabajos futuros

La implementación de la paralelización empleando OpenMP permitió que la ejecución del algoritmo fuera muchos más rápida, obteniéndose mejores resultados en menor tiempo. La paralelización con MPI permite ampliar la búsqueda en el espacio de soluciones, ya que al tener varias rutas se mejora la exploración de dicho espacio y es más probable que se llegue a una buena solución.

Además, con el uso de un clúster se obtiene un mayor poder de cómputo que permite evaluar un mayor número de rutas en menor tiempo. Se puede apreciar que la paralelización permitió reducir aproximadamente 23 veces el tiempo de ejecución del algoritmo; además, el implementar la metaheurística de algoritmos genéticos ayudó a encontrar mejores rutas que con el código secuencial. La hibridación de OpenMP y MPI es un enfoque interesante que merece mayor estudio.

Para aprovechar el poder de cómputo proporcionado por la grid Tarántula, donde se han implementado esquemas de comunicación para reducir el tráfico entre los nodos de la grid y se han propuesto esquemas de gridificación para diferentes algoritmos ([15], [16]), el siguiente paso de este trabajo es probar el efecto de la latencia cuando se comparten los nodos de diferentes clusters a través de la grid utilizando el enfoque de paso de mensajes.

Referencias

[1] R.Acosta, M. García-Ruiz, C.A. Banda. O.A. Barajas, J.M. Ramírez, P.D. Reyes y C.R. Bustos, *Implementación de un Cluster de alto rendimiento como herramienta*

para resolver problemas de cómputo científico, en Memorias de la 3a. Conf. Iberoamericana en Sistemas, Cibernética e Informática, CICSI 2004, (2004).

[2] A.G. Valdez y G. Campos, *Creación de un Cluster de Linux utilizando Knoppix*. Sociedad de la Información, No. 23 (2008).

[3] R. Rivera-López, A. Rodríguez-León, M. A. Cruz-Chávez y I. Y. Hernández-Baez, *Tarántula: Una grid de clusters de cómputo para el desarrollo de aplicaciones paralelas y en grid en México*, en Memorias del Coloquio de Investigación Multidisciplinaria, Orizaba, Ver. (2011).

[4] R. Martí, *Procedimientos Metaheurísticos de Optimización Combinatoria*, Matematiques, Vol. 1, No 1, pp. 3-62, 2003.

[5] K. Fujisawa, M. Kojima, A. Takeda, M. Yamashita, *Solving Large Scale Optimization Problems via Grid and Cluster Computing*, Journal of the Operations Research Society of Japan, 47(4), pp. 265-274, 2004.

[6] 6. M.A. Cruz-Chávez, A. Rodriguez-León, E.Y. Ávila-Melgar, F. Juárez-Pérez, J.C. Zavala-Díaz, R. Rivera-López, *Parallel Hybrid Evolutionary Algorithm in Grid Environment for the Job Shop Scheduling Problem*, Proc. of the 2nd. EELA-2 Conf., pp. 227-234, 2009.

[7] M.A. Cruz-Chávez, I. Hernández-Báez, A. Rodriguez-León, E.Y. Ávila-Melgar, F. Juárez-Pérez, F., A. Martínez-Oropeza, *PSAUPMP Application in Grid EELA-2, status 5: Parallel Simulated Annealing algorithm for the weighted Unrelated Parallel Machines Problem*, Second EELA-2 Grid School, Qro., Mex, (2009). http://applications.eu-eela.eu/app_list.php?l=20.

[8] B. Melián, J. A. Moreno Pérez, J. M. Moreno Vega, *Metaheuristics: A Global View. Inteligencia Artificial*, vol. 7, no. 19, (2003) pp. 7-28.

[9] M. Hahsler y K. Hornik, *TSP – Infrastructure for the Traveling Salesperson Problem*, Journal of Statistical Software, vol. 23, no. 2 (2007).

[10] D. S. Johnson y L.A. McGeoch, *The Traveling Salesman Problem: A Case Study in Local Optimization*, en Local Search in Combinatorial Optimization, E. H. L. Aarts and J. K. Lenstra (editors), Wiley, 1997, pp. 215-310.

[11] S. B. Liu, K. M. Ng, y H. L. Ong, *A New Heuristic Algorithm for the ClassicalSymmetric Traveling Salesman Problem*, en World Academy of Science, Engineering and Technology vol. 27, no. 48, (2007).

[12] M.A. Cruz-Chávez, A. Martínez-Oropeza y S.A. Serna Barquera, *Neighborhood Hybrid Structure for Discrete Optimization Problems*, en Memorias de 2010 Conf. of Electronics, Robotics and Automotive Mechanics (CERMA 2010), pp. 108-113.

[13] D.L. Applegate, R.E. Bixby, V. Chvátal y W.J. Cook, *The Traveling Salesman Problem, A Computational Study*, Princeton University Press, (2006).

[14] J. L. González Velarde y R. Z. Río Mercado, *Investigación de Operaciones en Acción: Aplicación del TSP en Problemas de Manufactura y Logística*, en Ingenierías, Vol. 11, No 4, Mayo, (1999), pp. 18-23.

[15] M. A. Cruz-Chávez, A. Rodríguez-León, E.Y. Ávila-Melgar, F. Juárez-Pérez, M.H. Cruz-Rosales, R. Rivera-López, *Gridification of Genetic Algorithm with Reduced Communication for the Job Shop Scheduling Problem*, International Journal of Grid and Distributed Computing, Vol. 3, No. 3, (2010) pp. 13-28 .

[16] A. Rodriguez-León, M. A. Cruz-Chávez, R. Rivera-López, E. Y. Ávila-Melgar, F. Juárez-Pérez y M.H. Cruz-Rosales, *A Communication Scheme for an Experimental Grid in the Resolution of VRPTW using an Evolutionary Algorithm*, en Memorias de 2010 Conf. of Electronics, Robotics and Automotive Mechanics (CERMA 2010), pp. 3-8.