ISSN 1870-4069

Implementación paralela de un algoritmo genético para el problema del agente viajero usando OpenMP

Edgar Martínez Vargas, Marcela Rivera Martínez, Luis René Marcial Castillo, Lourdes Sandoval Solis

Benemérita Universidad Autónoma de Puebla,
Facultad de Ciencias de la Computación, Puebla,
México

edgar-93@outlook.com, {cmr, lmc}@solarium.cs.buap.mx
maria.sandoval@correo.buap.mx

Resumen. En este trabajo se aborda el problema del agente viajero, resolviéndolo mediante la aplicación paralela de un algoritmo genético, implementado en lenguaje C, utilizando las directivas de OpenMP. Dichas soluciones al problema consisten en encontrar el camino más corto visitando todas las ciudades sin repetir alguna y con el menor costo posible. Las pruebas se llevaron a cabo en el Laboratorio Nacional de Supercómputo del Sureste de México y los resultados se comparan con problemas prueba reportados en la literatura.

Palabras clave: Algoritmos genéticos paralelos, agente viajero, OpenMP, lenguaje C.

Parallel Implementation of a Genetic Algorithm for the Traveling Salesman Problem Using OpenMP

Abstract. In this paper explain the traveling salesman problem, solving it through the parallel application of a genetic algorithm, implemented in C language, using OpenMP directives. These solutions to the problem consists to find the shortest route visiting all the cities without repeating some and with the lowest possible cost. The tests were performed at "Laboratorio Nacional de Supercómputo del Sureste de México" and the results are compared with benchmark problems reported in the literature.

Keywords: Parallel genetic algorithms, traveling salesman problem, OpenMP, C language.

1. Introducción

El problema del agente viajero es ampliamente estudiado ya que se considera como un problema difícil de resolver, denominándose en un lenguaje computacional como NP completo [1], es decir, es un problema para el que no se puede garantizar que se encontrará la mejor solución en un tiempo de cómputo razonable.

Dicho problema ha sido planteado a través de diversas técnicas, entre las que predomina la aplicación de algoritmos genéticos [2], este tipo de algoritmos son muy eficaces para problemas complejos de búsqueda y optimización, dado que se pueden adaptar de forma paralela para trabajos en los que se requiera manejar grandes volúmenes de datos. Algunos artículos que proponen soluciones al problema del agente viajero mediante algoritmos genéticos paralelos son los siguientes: GRISLAS [3], un algoritmo genético paralelo que combina los modelos de grillas e islas para encontrar buenas soluciones; y el modelo PPREGA (Parallel Pattern Reduction Enhanced Genetic Algorithm) [4], es un método escalable para reducir el tiempo de cálculo de los algoritmos genéticos.

También ha sido enfrentado mediante el algoritmo de recocido simulado [5], el objetivo general de este tipo de algoritmo es encontrar una buena aproximación al valor óptimo de una función en un espacio de búsqueda grande.

Otra técnica propuesta para resolver el problema es aplicando el algoritmo de colonia de hormigas [6], básicamente, este algoritmo menciona que las hormigas son capaces de seguir la ruta más corta en su camino de ida y vuelta a la colonia. Al desplazarse cada una va dejando un rastro de una sustancia química llamada feromona a lo largo del camino seguido, "transmitiéndose información" entre ellas.

En este trabajo se propone usar un algoritmo genético paralelo para encontrar soluciones a dicho problema, el cual se implementó en C utilizando las directivas de OpenMP [7] para paralelizarlo. Además, el propósito es reducir los tiempos de respuesta y encontrar soluciones buenas al aplicarlo en grandes problemas.

En la siguiente sección se presentan algunas notas preliminares sobre los orígenes y un diagrama general de los algoritmos genéticos, además de la definición del problema, en la sección 3 se explica brevemente el paradigma sobre los algoritmos genéticos paralelos, la sección 4 muestra y detalla el algoritmo propuesto, la sección 5 describe los experimentos y resultados de los mismos, así como del "speedup" y escalabilidad del algoritmo paralelo. La sección número 6 presenta las conclusiones y finalmente se listan las referencias utilizadas en el desarrollo de este trabajo.

2. Preliminares

2.1. Algoritmos genéticos

Los algoritmos genéticos fueron introducidos por John Holland a finales de los 60´s inspirándose en el proceso observado en la evolución natural de los seres vivos. Son algoritmos de búsqueda basados en la mecánica de la selección natural y en la genética. Estos combinan la supervivencia de los individuos más aptos entre las cadenas de estructuras con un intercambio aleatorio para formar un algoritmo de búsqueda [8]. Los

algoritmos genéticos están determinados por tres características fundamentales: selección de parejas, cruza y mutación. La figura 1 presenta el esquema general de un algoritmo genético.

2.2. Definición del problema

El problema del agente viajero está dado de la siguiente forma: Sean *n* ciudades de un territorio, se especifica una ciudad *k*, que será el inicio y fin del recorrido, el objetivo es encontrar una ruta que pase una sola vez por cada una de las ciudades y minimice el costo realizado por el viajero, es decir, se debe minimizar:

$$\min\left\{ z = \sum_{i=1}^{n} \sum_{j=1}^{n} C_{ij} x_{ij} \right\}, \tag{1}$$

donde C_{ij} es el costo asociado de viajar de la ciudad *i* a la ciudad *j*, y x_{ij} es la variable de decisión de visitar la ciudad *j* después de visitar la ciudad *i*. La variable de decisión es 1 en caso de que se visite la ciudad *j* después de visitar la ciudad *i* y 0 en caso contrario.

2.3. Algoritmo genético para el problema del agente viajero

En la figura 1 se presenta el diagrama de flujo de un algoritmo genético secuencial, el cual es la base para el algoritmo paralelo expuesto posteriormente.

Se programó un algoritmo genético secuencial para resolver el problema del agente viajero, de acuerdo al diagrama de la figura 1. Enseguida se describe brevemente dicha implementación.

La función de aptitud a utilizar es la dada en la ecuación (1), y cada cromosoma es una permutación de valores enteros entre 1 y la cantidad de nodos *n*. Por ejemplo, una representación posible dado 5 ciudades es: (1, 2, 3, 4, 5).

Generar la población inicial: En este caso se hizo de manera aleatoria.

Seleccionar parejas: La selección de parejas para este problema en especial es ruleta por costo, la cual consiste en crear una ruleta en la que cada cromosoma tiene asignada una fracción proporcional, dicha fracción proporcional es con respecto al costo de la función de aptitud.

Cruza: Para el desarrollo del problema se considera la reproducción o cruza por medio de 2 puntos esto quiere decir que se toman 2 puntos distintos en cada padre, y se intercambian los alelos entre los padres para generar el mismo número de individuos, tomando en cuenta los valores antes del punto de cruza del primer padre, los valores medios del segundo padre y los valores finales del primer padre, para la generación de uno de los hijos, para el otro es inverso. Dado que para el problema del agente viajero no debe haber ciudades repetidas, se utiliza la cruza denominada PMX (Partially Mappet Crossover) [9], donde para los valores que faltan en los hijos, se copian los valores de los padres de la siguiente forma: Si el valor a copiar no está en la subcadena,

copiarlo, en caso contrario, se copia por el valor que tenga dicha subcadena en el otro padre.

Mutación: Busca cambiar aleatoriamente la información genética. La mutación que se utiliza es mutación por inversión, en ésta se generan dos números diferentes aleatorios entre 1 y el tamaño del cromosoma, esta subcadena se invierte posteriormente.

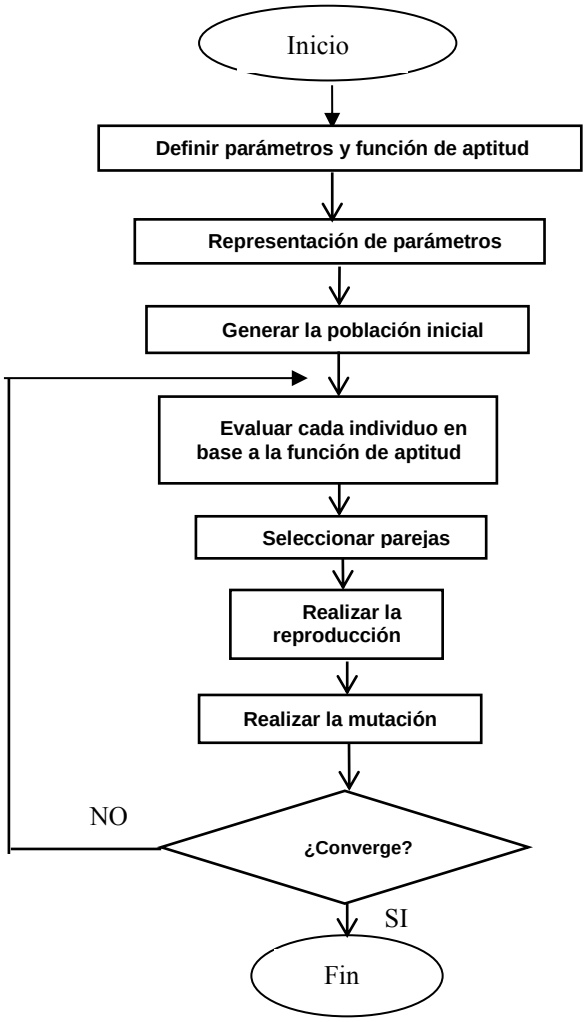

Fig. 1. Diagrama del funcionamiento de un algoritmo.genético.

3. Algoritmos genéticos paralelos

Las técnicas de programación de alto rendimiento (performance), más comúnmente llamadas técnicas de procesamiento paralelo o técnicas de paralelismo, constituyen un

conjunto de métodos computacionales utilizados para la resolución de problemas sobre sistemas con una capacidad de procesamiento superior al tradicional modelo de computadora de Von Neumann. Estos sistemas, conocidos como multiprocesadores, permiten abordar la resolución computacional de problemas demasiado complicados de acuerdo a la complejidad computacional inherente al propio problema, de acuerdo a la dimensión de las instancias consideradas, o debido a que trabajan sobre un enorme volumen de datos que serían virtualmente imposibles de resolver mediante el paradigma tradicional. La característica principal de los sistemas multiprocesadores consiste en disponer de un conjunto de unidades de procesamiento interconectadas por algún medio que posibilita la comunicación de datos y de control entre ellos [10].

En la actualidad, el continuo aumento en el tamaño de los problemas ha encaminado a proponer distintas opciones a los enfoques tradicionales de programación para la resolución de problemas complejos. En este contexto, varias heurísticas se han aplicado a los problemas relacionados con búsqueda y optimización. Las técnicas de computación evolutiva, y los algoritmos genéticos en particular, se han manifestado como métodos propicios, ya que permiten un diseño paralelizable fácilmente, para resolver los problemas complejos, tal como es el problema del agente viajero.

3.1. Clasificación

A continuación se presentan las 3 maneras básicas para clasificar o paralelizar un algoritmo genético.

1. *Maestro-esclavo*: Estos algoritmos trabajan con una única población de individuos que será gestionada por el nodo maestro. La evaluación de la función de los individuos y/o la aplicación de los operadores genéticos puede ser realizada por los nodos esclavos.
2. *Grano fino*: Este tipo de algoritmos han sido diseñados para ser implementados usando computadores masivamente paralelos. Ahora, la población se encuentra dividida espacialmente entre los distintos procesadores e, idealmente, cada procesador debería albergar un único individuo. El cruce y la selección de individuos se hará entre individuos que pertenezcan a un mismo vecindario.
3. *Grano grueso*: La característica más importante de estos algoritmos es el uso de múltiples poblaciones y la inclusión de un nuevo operador denominado migración; la migración de individuos se debe aplicar entre las distintas poblaciones. Dado que cada una de las poblaciones evolucionan independientemente, el porcentaje de migración será muy importante de cara a obtener resultados satisfactorios.

3.2. Paradigmas

A su vez, este tipo de clasificación se puede llevar a cabo siguiendo tres estilos distintos de programación:

1. *Paralelismo en datos*: El compilador se encarga de la distribución de los datos guiado por un conjunto de directivas que introduce el programador. Para este

modelo de paralelismo destacan los lenguajes HPF (High Performance Fortran) y OpenMP.

2. *Programación por paso de mensajes*: Es el método más utilizado para programar sistemas de memoria distribuida. La forma básica consiste en que los procesos coordinan sus actividades mediante el envío y la recepción de mensajes. Las librerías más utilizadas son el estándar MPI (Message Passing Interface) y PVM (Parallel Virtual Machine).
3. *Programación por paso de datos*: Para este caso la transferencia de datos entre los procesadores se realiza con primitivas tipo put-get, lo que evita la necesidad de sincronización entre los procesadores emisor y receptor.

4. Algoritmo propuesto

El algoritmo que a continuación se propone se basa en la configuración básica de maestro-esclavo, ya que existe un hilo maestro que asigna segmentos del algoritmo, a otros hilos para ejecutarse de manera paralela, cuando los hilos esclavos terminen de ejecutar el fragmento del algoritmo entonces se devuelven los datos al hilo maestro para que éste continúe ejecutando el programa. Se implementa a través del paradigma o modelo de programación de paralelismo en datos; la ventaja o aprovechamiento de realizar esta composición está dada en el tiempo de ejecución, además de que es fácil de implementar.

Enseguida se explican los segmentos (operadores) del algoritmo que se paralelizan, para esto se retoma el diagrama mostrado en la figura 1.

Se inicia generando y evaluando a la población en paralelo, seguidamente se seleccionan las parejas de manera secuencial, éste operador no se puede aplicar en paralelo ya que existe dependencia de datos. Posteriormente, se cruzan las parejas seleccionadas para formar nuevos cromosomas (hijos), este operador se trabaja sobre un 50 por ciento de la población en curso. También se aplica el operador de mutación sobre la población en curso, el porcentaje de la mutación se pasa como parámetro al inicio de la ejecución del programa. Estos dos últimos operadores se realizan de forma paralela. Y, finalmente, se actualiza la población para encontrar la mejor solución.

```
Inicio
  Hacer en paralelo
    Generar la población inicial
    Calcular la evaluación
  Fin paralelo
  Ordenar la población (menor a mayor)
  Mientras no se cumpla la condición de terminación
    Seleccionar parejas
    Hacer en paralelo
      Cruzar parejas (50% de la población en curso)
      Mutar a la población en curso
      Actualizar población
    Fin paralelo
  Fin mientras
Fin
```

Algoritmo 1. Algoritmo genético paralelo para resolver el problema del agente viajero.

Las directivas de OpenMP que se utilizaron para implementar el algoritmo son las siguientes:

- #pragma omp sections [cláusulas]
  Esta es una directiva que paraleliza trabajo no iterativo, es decir, paraleliza regiones o secciones de código que son independientes y NO participa ningún bucle. Para este caso se utiliza para la declaración de las matrices (Población inicial, población de los hijos) y para la declaración de otras estructuras de datos que se requieren para el programa.
- #pragma omp for [cláusulas]
  Esta directiva sirve para subdividir el ciclo for para que cada hilo trabaje por separado las iteraciones de éste, no funciona en el bucle "while". La mayor parte de la paralelización del algoritmo utiliza esta directiva; se emplea para generar y evaluar a la población, en los operadores de cruza y mutación. Aunque parezca una directiva muy fácil de usar, se recomienda tener mucho cuidado al momento de manipular las variables que se pasen dentro de las cláusulas, ya que puede haber casos en los que la variable es global y no importa que la modifiquen todos los hilos o puede ser el caso que cuando una variable se modifique no le afecte al trabajo de los demás hilos. En pocas palabras, se debe tener cuidado con la dependencia de datos.

5. Experimentos y resultados

El algoritmo propuesto se validó utilizando problemas prueba reportados en la literatura [11], en la supercomputadora del Laboratorio Nacional de Supercómputo del Sureste de México [12], usando el lenguaje de programación C [13].

Para cada problema prueba se realizaron 15 ejecuciones del algoritmo con distintos parámetros sobre la cantidad de generaciones, tamaño de la población, porcentaje de mutación y la cantidad de hilos.

La tabla 1 reporta los costos menores obtenidos con las ejecuciones en el algoritmo genético secuencial y el algoritmo genético paralelo, así como el mejor reportado en la literatura en la actualidad.

La tabla 2 muestra una comparación de algunos tiempos de ejecución para cada problema variando la cantidad de hilos. El tiempo reportado esta dado en segundos. Para los problemas de 10 y 20 ciudades se aplicaron los siguientes parámetros: 100 generaciones, 100 individuos y 0.3 porcentaje de mutación. Para los problemas restantes se aplicaron los siguientes parámetros: 500 generaciones, 500 individuos y 0.5 porcentaje de mutación, se trabaja con éstos parámetros con el propósito de comparar los tiempos de ejecución al operar con grandes volúmenes de datos. La última columna de la tabla 2 presenta la cantidad de hilos, realizando pruebas inicialmente con 2 y hasta 16 hilos.

Las figuras 2, 3 y 4 que se presentan son el resultado de hacer pruebas con el problema tsplib_problem_d1291, 1291 ciudades, con los siguientes parámetros: 100 generaciones, 100 individuos de población, 0.3 de porcentaje de mutación y 16 hilos de ejecución.

Tabla 1. Costos obtenidos del algoritmo genético secuencial, paralelo y los reportados en la literatura.

Problema	Cantidad Ciudades	AG secuencial	AG paralelo	Mejor reportado
TSP_small10	10	**1324.55**	**1324.55**	**1324.554**
TSP_small20	20	1847.55	**1836.92**	**1836.924**
data_100	100	**3746.40**	**3746.40**	**3746.40**
tsplib_problem_a280	280	4271.11	2828.80	**2579**
data_500	500	8862	**8456**	**8456**
tsplib_problem_d1291	1291	153672	98940	**50801**

Tabla 2. Comparación de los tiempos de ejecución del algoritmo genético secuencial y el paralelo.

Problema	Cantidad Ciudades	Tiempo AG secuencial	Tiempo AG paralelo	Cantidad Hilos
TSP_small10	10	0.0130	0.0134	2
TSP_small10	10	-	0.0255	5
TSP_small10	10	-	21.32	16
TSP_small20	20	0.0219	0.0287	2
TSP_small20	20	-	0.0317	5
TSP_small20	20	-	21.17	16
data_100	100	2.61	3.15	2
data_100	100	-	2.02	5
data_100	100	-	1.98	10
tsplib_problem_a280	280	13.42	6.11	5
tsplib_problem_a280	280	-	5.42	10
tsplib_problem_a280	280	-	37.31	16
data_500	500	36.78	25.81	2
data_500	500	-	12.11	10
data_500	500	-	28.08	16
tsplib_problem_d1291	1291	266.15	131.63	2
tsplib_problem_d1291	1291	-	77.18	5
tsplib_problem_d1291	1291	-	52.80	10
tsplib_problem_d1291	1291	-	93.40	16

La figura 2 representa los tiempos de ejecución del programa paralelo, el eje vertical simboliza el tiempo que esta dado en segundos y el eje horizontal indica el número de procesadores en los que se ejecutó el programa.

El factor de mejora del rendimiento o también conocido como "speedup" se define como:

$$S(n) = T(1) / T(n), \tag{2}$$

donde *T(1)* es el tiempo de ejecución del algoritmo secuencial (1 procesador) y *T(n)* es el tiempo de ejecución del algoritmo paralelo ejecutado sobre *n* procesadores. La figura 3 muestra el comportamiento de la función del speedup.

Por último, la figura 4 representa la escalabilidad del programa paralelo y su función está dada de la siguiente manera:

$$E(n) = S(n) / n = T(1) / (n*T(n)). \tag{3}$$

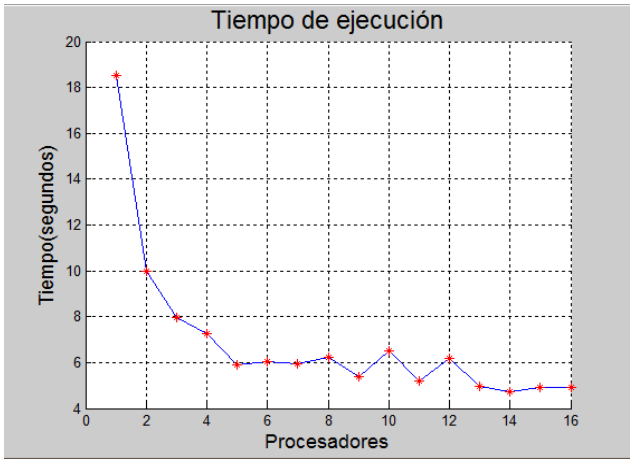

Fig. 2.Tiempos de ejecución del algoritmo genético paralelo para el problema tsplib_problem_d1291.

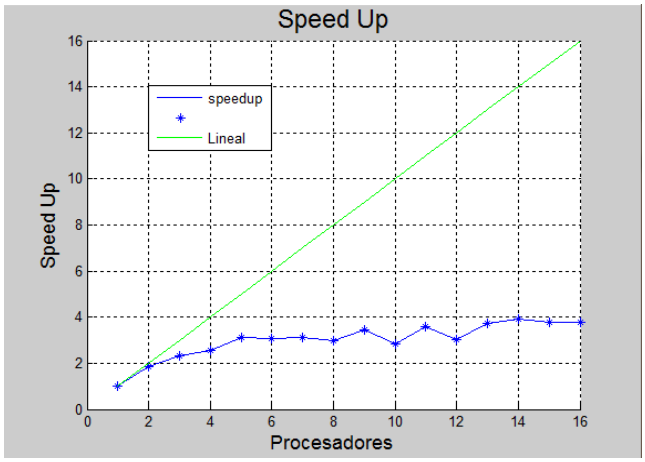

Fig. 3. Speedup del algoritmo genético paralelo para el problema tsplib_problem_d1291.

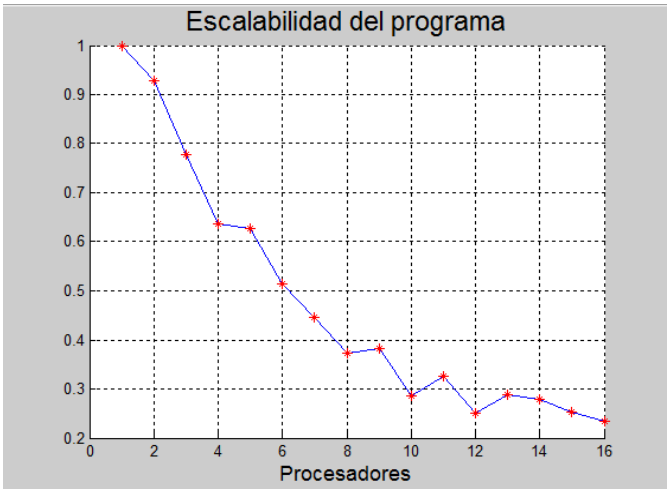

Fig. 4. Escalabilidad del algoritmo genético paralelo para el problema tsplib_problem_d1291.

6. Conclusiones

Los algoritmos genéticos tradicionales son de gran ayuda para problemas de búsqueda y optimización, además, estos algoritmos tienen la gran ventaja de ser propicios para poder paralelizarlos y así poder aprovechar o tener un mejor rendimiento sobre nuestro equipo de cómputo.

Se observa que con el algoritmo paralelo se alcanzan mejores resultados en comparación con el algoritmo secuencial, así mismo se obtiene un mejor rendimiento en cuestiones de tiempo de ejecución, logrando un porcentaje de mejora de tiempo de 24% para el caso de 100 ciudades y un 80% de mejora para el problema de 1291 ciudades. Con respecto a la cantidad de hilos, se nota que para los problemas de 10 y 20 ciudades es suficiente ejecutar el algoritmo con 2 hilos, mientras que para los problemas de igual o mayor a 100 ciudades se trabaja mejor con 10 hilos. Se concluye que es factible aplicar el algoritmo paralelo para problemas con cantidad de ciudades grandes.

El próximo objetivo a lograr en un futuro cercano es mejorar el algoritmo genético paralelo con la finalidad de encontrar mejores soluciones. Por ejemplo, aplicando distintas técnicas de mutación para explorar en diferentes espacios de búsqueda y generar nuevos individuos después de cada generación para sustituir a los peores individuos de la población.

Agradecimientos. Agradecemos el apoyo financiero de la Vicerrectoría de Investigación y Estudios de Posgrado de la Benemérita Universidad Autónoma de Puebla a través del proyecto SASM-ING16-G, de la misma manera agradecemos al Laboratorio Nacional de Supercómputo del Sureste de México por las facilidades prestadas para la realización de las pruebas.

Referencias

1. Dasgupta, S., Papadimitriou, C., Vazirani, U.: NP-complete problems: Algorithms. McGraw-Hill Education, pp. 233–237 (2006)
2. Saloni, G., Poonam, P.: International Journal of Advanced Research in Computer Science and Software Engineering, Vol.3 (2013)
3. Poveda, R., Gómez, J., León, E.: Un algoritmo genético paralelo que combina los modelos de grillas e islas para encontrar soluciones óptimas cercanas al problema del agente viajero. Vol. 5, No. 3, pp. 13–19 (2008)
4. Tsai, C., Tseng, S., Chiang, M., Yang, C.: A Fast Parallel Genetic Algorithm for Traveling Salesman Problem. pp. 241–250 (2010)
5. Fang, Y.: Solving Traveling Salesman Problem Using Parallel Genetic Algorithm and Simulated Annealing. pp. 2 (2010)
6. Dorigo, M., Gambardella, L.: Ant colony system: a cooperative learning approach to the traveling salesman problem. Vol. 1, pp. 53–66 (2002)
7. OpenMP. http://openmp.org/wp (2016)
8. Haup, R. L., Haup, S. E.: Practical Genetic Algorithms. John Wiley & Sons, Inc. (1998)
9. Golberg, D., Lingles, R.: The travelling salesman problem. In: Proceedings of the First International Conference on Genetic Algorithms (1986)
10. Nesmachnow, S.: Algoritmos genéticos paralelos y su aplicación al diseño de redes de comunicaciones confiables. pp. 30 (2004)
11. Problemas benchmark. https://sites.google.com/site/logisticslaboratory/lecture/ tsp-program-benchmark-problems (2016)
12. LNS. http://www.lns.buap.mx (2016)
13. Lenguaje C. http://www.cprogramming.com (2016)