Un algoritmo memético paralelo para TSP

Joel Artemio Morales Viscaya
División de Estudios de Posgrado e Investigación
Instituto Tecnológico de La Paz
La Paz, B.C.S., México
Email: iscviscaya@gmail.com

Marco Antonio Castro Liera
División de Estudios de Posgrado e Investigación
Instituto Tecnológico de La Paz
La Paz, B.C.S., México
Email: mcastro@itlp.edu.mx

***Resumen*—Se presenta un nuevo algoritmo memético aplicado al problema del agente viajero, con una estrategia de búsqueda local que mezcla características de 3-OPT y Lin-Kernighan. El algoritmo se implementó en un cluster de computadoras y se incluyen resultados estadísticos de su desempeño al aplicarlo en diferentes instancias obtenidas de la TSPLIB de entre 100 y 300 ciudades. Además se demuestra su rendimiento superior al de otras heurísticas como ACO, AG, 2-OPT y 3-OPT.**

I. Introducción

El problema del agente viajero simétrico (TSP por sus siglas en inglés, Traveling Salesman Problem) es un problema clásico y ampliamente estudiado de optimización combinatoria que consiste en encontrar, dado un conjunto de ciudades, la ruta más corta que permita a un agente viajero visitar todas ellas y regresar a la ciudad en la que inició el recorrido, en donde la distancia entre las ciudades debe ser simétrica; es decir, la distancia entre la ciudad A y la ciudad B debe ser la misma que entre B y A para cualquier par de ciudades en el conjunto.

TSP pertenece a la clase de problemas de optimización NP-duro (es un problema duro dentro de los no polinomiales) de acuerdo a [1] y es importante, pues constituye un componente central de un número considerable de problemas del mundo real. Sus aplicaciones incluyen pero no están restringidas a la logística, la fabricación de circuitos integrados y tableros de circuitos perforados, medición por rayos x, data clustering, mira de telescopios, análisis de secuencias genéticas, guiado de láser para cristal, conexión de antenas, construcción, programación de tareas, la atención a llamadas de emergencia, servicio postal, entre otras.

TSP es posiblemente el problema más famoso y extensamente estudiado en el campo de la optimización combinatoria [2]. En años recientes, muchas heurísticas o meta-heurísticas han sido aplicadas para resolver problemas NP-duros (ya que por definición no existen algoritmos que los resuelvan en tiempos razonables), como por ejemplo el recocido simulado [3], la Búsqueda Tabú (TS) [4], Algoritmos Genéticos[5], Redes Neuronales (RN) [6], Optimización por Colonia de Hormigas (ACO) [7], Optimización por enjambre de partículas (PSO) [8], entre otras.

Los algoritmos existentes para resolver TSP se pueden dividir en dos clases: algoritmos exactos y algoritmos de aproximación. Los algoritmos exactos pueden garantizar que convergen a la solución óptima en una cantidad acotada de pasos. De momento, el algoritmo exacto más eficiente está basado en el corte de planos y técnicas de programación lineal [9] y ha sido aplicado a instancias grandes de TSP. Sin embargo, estos algoritmos exactos tienen una complejidad computacional muy elevada [10], además de ser muy difíciles de programar y requerir grandes cantidades de almacenamiento en memoria. En contraste, los algoritmos de aproximación obtienen buenas soluciones en tiempos razonables, pero no pueden garantizar que van a converger a soluciones óptimas. Los algoritmos de aproximación para TSP se pueden dividir a su vez en dos categorías: algoritmos de construcción de rutas [11] y algoritmos de mejoramiento de rutas [12].

Los algoritmos de construcción de rutas van construyendo la ruta (o tour) de manera gradual, añadiendo una nueva ciudad en cada paso, mientras que los algoritmos de mejoramiento de rutas empiezan con una solución inicial y tratan de mejorarla en cada paso, realizando intercambios.

II. Algoritmos meméticos

Los algoritmos evolutivos (EA) pertenecen a una rama de la inteligencia artificial que involucra problemas de optimización y se inspiran en los mecanismos de la evolución biológica. Son un tipo de estrategia de búsqueda global [13] que se ha utilizado con éxito para resolver múltiples problemas [14]. Sin embargo, en particular para problemas combinatorios complejos se considera que las estrategias puras de este tipo, como son los algoritmos genéticos, no son suficientemente buenas [15].

Algoritmo 1 Estructura general de un algoritmo memético

```
1:  t=0
2:  P(t)=InicializarLaPoblación
3:  P(t)=BúsquedaLocal(P(t))
4:  P(t)=Selección(P(t))
5:  mientras criterio de parada no cumplido haz
6:      P(t)=Cruza(P(t))
7:      P(t)=Mutación(P(t))
8:      P(t)=BusquedaLocal(P(t))
9:      P(t+1)=Seleccion(P(t))
10:     t=t+1
11: fin mientras
```

La combinación de algoritmos evolutivos poblacionales con heurísticas de búsqueda local forman lo que se conoce como un algoritmo memético (MA). Debido a la mezcla de las

ISBN: 978-607-97128-2-2

propiedades incluyentes de los EA y la exploración a detalle de zonas prometedoras de la búsqueda local, los MA han mostrado ser más eficientes y eficaces en la resolución de problemas combinatorios [16], [17]

El pseudocódigo general de un MA basado en un AG se puede observar en el algoritmo 1.

III. Algoritmos de búsqueda local

Un enfoque heurístico para los problemas de optimización combinatoria es la mejora iterativa de las soluciones, mediante una transformación que tome soluciones del problema y las lleve a nuevas soluciones con mejor aptitud, hasta que ya no sea posible encontrar dichas transformaciones.

Para TSP se han estudiado algoritmos de búsqueda local conocidos como k-opt, cuya idea central es tomar un recorrido hamiltoniano (un tour que visita todas las ciudades y regresa a la inicial) y llevar a cabo el reemplazo de k aristas o caminos en el recorrido por k aristas que no se encuentren en el mismo con la condición de seguir formando un recorrido hamiltoniano, de acuerdo a [13] se considera que para $k > 4$ son demasiado costosas.

IV. Clúster de computadoras

Una alternativa para disminuir el tiempo de ejecución en la resolución de problemas complejos de ingeniería es el cómputo paralelo, que consiste en la ejecución de múltiples instrucciones de un programa simultáneamente en varias unidades de procesamiento [18], [19], una forma de implementar el cómputo paralelo son los clústers de computadoras, que consisten en un conjunto de equipos de cómputo interconectados que colaborativamente intercambian y procesan información [20]. Sólo los procesadores locales tienen acceso directamente a la memoria local en cada nodo, por tanto, si un nodo requiere acceder a la memoria de otro nodo, se hace con un modelo de comunicación de paso de mensajes. Dentro del conjunto de nodos a uno de ellos se le asigna la tarea de iniciar, distribuir tareas y recopilar resultados de ejecución de la aplicación paralela, este nodo se denomina maestro [20], como se muestra en la Fig. 1

MPI es una estandarizacióon de las librerías de paso de mensajes que fue creada por el MPI-FORUM, es portable y tiene librerías para ser implementado en lenguage C, C++, Fortran, Java, Phyton y Perl, entre otros.

Existen varias implementaciones del estándar, algunas de ellas de software libre como OpenMPI.

V. Estrategia propuesta

V-A. Estructura

Se propone un algoritmo que se puede dividir en tres fases, la inicialización de los procesos, uno maestro y los restantes esclavos. El núcleo o la optimización, que es un algoritmo memético basado en un algoritmo genético de subpoblaciones estáticas con migración y una última fase en que cada proceso esclavo envía su mejor resultado al proceso maestro, para que encuentre el mejor de todos.

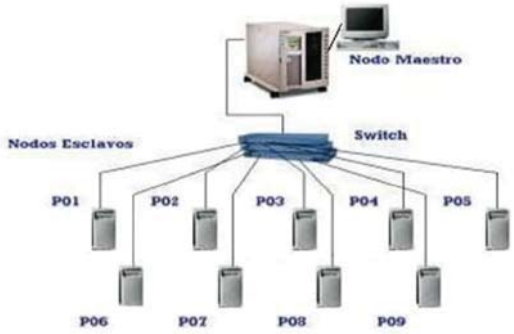

Figura 1. Esquema de un clúster de computadoras

Los procesos esclavos actúan sobre cada una de las subpoblaciones. Una vez que se cumple un cierto número de generaciones en el algoritmo memético, cada nodo esclavo comparte información con los demás en un símil a la migración. Esta estrategia es lo que se conoce en AG como modelo de islas y ha demostrado producir mejores soluciones que poblaciones aisladas o una sola población masiva [21].

Una vez que se cumple la condición de parada (el número de generaciones o vueltas del algoritmo) cada nodo esclavo envía sus resultados hacia el nodo maestro.

V-B. Algoritmo memético

Para garantizar una alta diversidad en las soluciones iniciales de nuestro algoritmo memético, empezamos generando una población inicial de soluciones de forma aleatoria. Sobre dichas soluciones se aplica el algoritmo de búsqueda local.

Un AG consta de tres operaciones básicas que se conocen como operadores genéticos: selección, cruza y mutación. Selección es el proceso por el cuál se elige que individuos o soluciones permaneceran en la generación siguiente. Decidimos utilizar elitismo, que consiste en simplemente conservar a los mejores individuos de la población.

Cruza es como se le conoce al proceso que genera nuevas soluciones utilizando los individuos que superaron la etapa de selección, se decidió utilizar un algoritmo conocido como Order Crossover (OX), propuesto por Davis [22] para permutaciones.

La mutación se considera como un operador secundario en los algoritmos genéticos. Se aplica solamente a una cantidad reducida de los individuos y produce cambios aleatorios grandes en sus elementos.

Además, cada cierto número de generaciones se envía una cantidad prefijada de individiduos (los mejores) hacia otra subpoblación. Esta operación se conoce como migración.

Se modificó el ciclo principal de un AG, agregando después de los operadores genéticos de cruza y mutación el algoritmo de búsqueda local, como se puede observar en la Fig. 2

V-C. Búsqueda local

Debido a que las estrategias puras 2-opt y 3-opt mejoran las soluciones de manera muy lenta y que la más utilizada Lin-Kernighan (LK), basada en determinar cuántas aristas

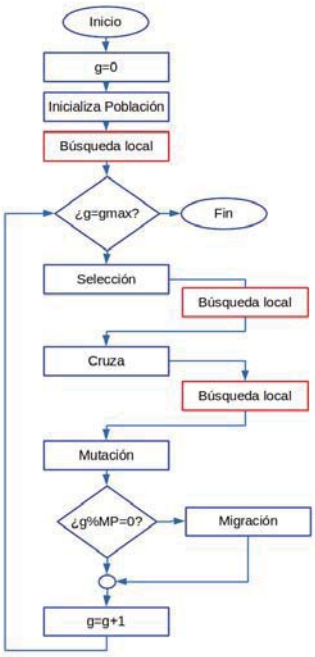

Figura 2. Diagrama del algoritmo memético propuesto

cambiar en cada iteración es computacionalmente demasiado costosa y difícil de programar, además de requerir mucho almacenamiento en memoria, se propone un algoritmo de mejoramiento de rutas nuevo (búsqueda local) basado en 3-opt y en la estrategia LK.

Ya que se deben visitar todas las ciudades, TSP se puede reducir a encontrar el orden en que deben visitarse las mismas, lo cuál hace que la manera más adecuada de representar soluciones sea como un vector de n elementos, con n igual al número de ciudades de la instancia.

Sin embargo, para aplicar los algoritmos de búsqueda local es más adecuado representar el problema en términos de grafos, vértices y aristas, por lo que es necesario primeramente traducir estas soluciones a un conjunto de aristas.

El algoritmo que se desarrolló se basa en hacer reemplazos de tres aristas, pero eligiendo qué aristas se van a reemplazar y, por cuáles aristas van a ser reemplazas de manera "inteligente", de manera similar a cómo ocurre en LK. A grandes razgos, el algoritmo consta de los siguientes pasos:

1. Se elige al azar un nodo n_1 contenido en la ruta original.
2. Se selecciona cualquier arista x_1 adyacente a n_1.
3. Se elige una arista y_1 que conecte el otro extremo de n_1 tal que y_1 tenga un costo menor a x_1. Si no es posible encontrar y_1, se regresa a 2 y se selecciona la otra arista adyacente a n_1, si tampoco se puede encontrar y_1 se selecciona otro nodo n_1.
4. Se elige una arista x_2 que contiene al extremo de y_1 que no está en x_1, dando prioridad a la de costo mayor.
5. Se elige de manera aleatoria x_3, cómo cualquier arista contenida en la ruta original distinta de x_1 y x_2.
6. Se eligen y_2, y_3 tales que al reemplazar con x_1,x_2,x_3 por y_1,y_2,y_3 el recorrido resultante sea hamiltoniano (pueden no ser únicas).
7. Si algún intercambio encontrado en 6 produce un costo menor al inicial, se efectúa el reemplazo y se regresa al paso 1.
8. En caso contrario, si ningun par y_2, y_3 produce un recorrido mejor, regresamos a 5 y seleccionamos otra x_3.
9. En caso de haber probado todos los nodos x_3 posibles, se regresa al paso 4 y se elige una nueva x_2.
10. En caso de haber probado todos los x_2 posibles, se regresa al paso 3 seleccionando una arista y_1 distinta. Si ya se probó con todas las aristas y_1, entonces se ha encontrado un óptimo local y el algoritmo termina.

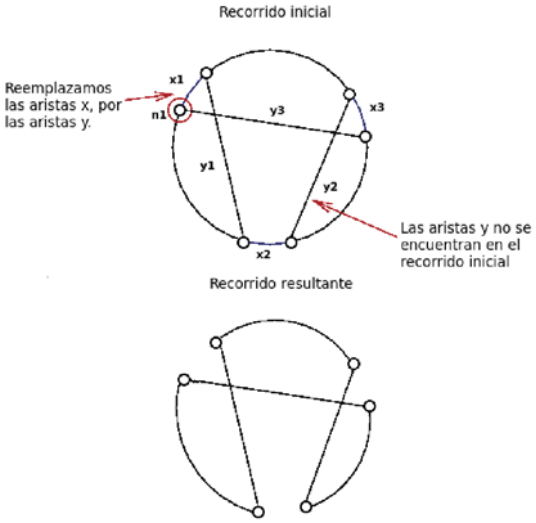

Figura 3. Ejemplo de iteración posible con reemplaza de tres aristas

Una posible iteración se puede observar en la Fig. 3

VI. Implementación y pruebas

Para el desarrollo del algoritmo memético se utilizó lenguaje C, compilado con GCC 4.8.4, un conjunto libre de compiladores distribuidos bajo la licencia GPL, se utilizó la librería OpenMPI en su versión 1.6.4 para el paso de mensajes entre los procesos, sobre Linux Mint 17.3 de 64 bits.

Se eligieron 20 instancias de tamaño mediano (entre 100 y 300 ciudades) de TSP para llevar a cabo las pruebas, dichas instancias se obtuvieron de la librería TSPLIB creada por el alemán G. Reinelt [23].

Las pruebas fueron llevadas a cabo en un clúster de 4 computadoras con procesadores i7-3770 con 8 GB de memoria RAM.

Se decidió hacer la comparación contra las estrategias heurística más utilizadas para resolver este problema, la optimización por colonia de hormigas, concebida originalmente para resolver TSP y un AG puro, el más utilizado de todos los algoritmos heurísticos de optimización. Debido a la naturaleza estocástica de los algoritmos heurísticos, se decidió utilizar una

muestra de 100 ejecuciones con cada una de las estrategias heurísticas.

De igual forma se comparó el desempeño de la estrategia memética con el algoritmo de búsqueda local desarrollado, contra el desempeño de la estrategia utilizando 2-opt y 3-opt.

VII. Resultados experimentales

En la tabla I se presenta una comparación entre el algoritmo de búsqueda local propuesta, contra una versión acelerada de 2-opt y contra 3-opt al utilizarlos para aproximar soluciones de la instancia de 225 ciudades propuestas por Reinelt de la TSPLIB, cuyo tour óptimo tiene una distancia de 3919.

Tabla I
Comparación de algoritmos de búsqueda local

Algoritmo	Distancia	Varianza	Tiempo	ERP
2-opt	4319.47	75.79	1.16s	10.21 %
3-opt	4053.39	33.57	8.27s	3.42 %
Propuesta	4044.68	45.64	0.60s	3.20 %

En la tabla II se muestra una comparación entre sistema de hormigas (AS), un algoritmo genético (AG) y el algoritmo memético propuesto (AM) al resolver la instancia de 100 ciudades propuestas por Krolak/Felts/Nelson de la TSPLIB. Dicha instancia tiene un tour óptimo con un costo de 21282. Las tablas III, IV y V muestran los parámetros para cada heurística.

Tabla II
Comparación de Heurísticas

Algoritmo	Distancia	Tiempo	ERP
AS	25049	21.26s	17.7 %
AG	21925	13.69s	3.02 %
memético	21282	1.02s	0.0 %

Tabla III
Parámetros del Sistema de Hormigas Paralelo

Parámetro	Valor
Colonias	32
Iteraciones	100
Tamaño de la Colonia	1000
Factor de evaporación	0.1

Tabla IV
Parámetros del AG Distribuido

Parámetro	Valor
Subpoblaciones	32
Generaciones	5000
Población	1024
Probabilidad de Selección	50 %
Probabilidad de Mutación	20 %
Elitismo	10
Período Migratorio	50
Tasa de Migración	5

Tabla V
Parámetros del algoritmo memético

Parámetro	Valor
Subpoblaciones	32
Generaciones	15
Población	10
Probabilidad de Selección	40 %
Probabilidad de Mutación	1 %
Elitismo	1
Período Migratorio	3
Tasa de Migración	1

Las tablas VI, VII y VIII muestran los resultados obtenidos con el algoritmo propuesto para diferentes instancias del problema tomadas de TSPLIB. El único parámetro que se modifica en cada caso es el tamaño de la población, el cual se muestra en cada tabla.

Tabla VI
Resultados experimentales instancias Churritz y Reinelt (TSP225).

Instancia	Óptimo	Población	ERP	Tiempo
ch130	6110	13	0.0 %	3.2s
ch150	6528	15	0.0 %	3.3s
tsp225	3919	22	0.0 %	21.3s

Tabla VII
Resultados experimentales instancias Krolak/Felts/Nelson.

Instancia	Óptimo	Población	ERP	Tiempo
kroA100	21282	10	0.0 %	1.0s
kroB100	22141	10	0.0 %	1.0s
kroC100	20749	10	0.0 %	1.0s
kroD100	21294	10	0.0 %	1.1s
kroE100	22068	10	0.0 %	1.0s
kroA150	26524	15	0.0 %	4.2s
kroB150	26130	15	0.0 %	4.4s
kroA200	29368	20	0.0 %	15.7s
kroB200	29437	20	0.0 %	14.6s

Tabla VIII
Resultados experimentales instancias Padberg/Rinaldi.

Instancia	Óptimo	Población	ERP	Tiempo
pr107	44303	10	0.0 %	3.0s
pr124	59030	12	0.0 %	3.3s
pr136	96772	13	0.0 %	3.1s
pr144	58537	14	0.0 %	9.0s
pr152	73682	15	0.0 %	8.8s
pr226	80369	22	0.0 %	44.3s
pr264	49135	26	0.0 %	94.7s
pr299	48191	30	0.0 %	75.5s

VIII. Conclusiones y trabajo futuro

El algoritmo de búsqueda propuesto parece ser una mejor alternativo al algoritmo 3-opt y a la versión favorable de 2-opt ya que, de acuerdo a la tabla I, la media del costo de la

ISBN: 978-607-97128-2-2

ruta encontrada por dicho algoritmo es claramente mejor que la obtenida utilizando 2-opt y si bien la media es similar a la encontrada por 3-opt, el algoritmo propuesto es hasta diez veces más rápido y, por tanto más adecuado para llevarse a cabo masivamente en un algoritmo evolutivo poblacional.

El algoritmo memético basado en la estrategia de búsqueda local propuesta encuentra el recorrido óptimo de las instancias de la TSPLIB de hasta 300 ciudades utilizadas como prueba de acuerdo a las tablas VI, VII y VIII. Por otro lado, utilizando un sistema de hormigas paralelo o un algoritmo genético distribuido, incluso para las instancias de 100 ciudades propuestas por Krolak, Felts y Nelson, no se alcanza el óptimo aún en tiempos diez veces mayores que el requerido por el algoritmo memético de acuerdo a la tabla II.

Actualmente se está trabajando en la implementación del algoritmo memético en una arquitectura paralela distinta como son las GPU, que recientemente han obtenido resultados mejores en relación a los *cluster* en algunos algoritmos paralelizables.

Otra cuestión que queda como trabajo futuro es la comparación de la heurística de búsqueda local propuesta contra estrategias más elaboradas presentes en la literatura como la de Lin-Kernighan-Helsgaun, entre otras.

Referencias

[1] S. Arora, "Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems," *Journal of the ACM*, vol. 45, no. 5, pp. 753–782, 1998.

[2] G. Gutin and A. P. Punnen, *The traveling salesman problem and its variations*, vol. 12. Kluwer Academic Publishers, 2002.

[3] Y. Chen and P. Zhang, "Optimized annealing of traveling salesman problem from the nth-nearest-neighbor distribution," *Physica A: Statistical Mechanics and its Applications*, vol. 371, no. 2, pp. 627–632, 2006.

[4] J.-q. Li, Q.-k. Pan, and Y.-C. Liang, "An effective hybrid tabu search algorithm for multi-objective flexible job-shop scheduling problems," *Computers & Industrial Engineering*, vol. 59, no. 4, pp. 647–662, 2010.

[5] S. J. Louis and G. Li, "Case injected genetic algorithms for traveling salesman problems," *Information sciences*, vol. 122, no. 2, pp. 201–225, 2000.

[6] K.-S. Leung, H.-D. Jin, and Z.-B. Xu, "An expanding self-organizing neural network for the traveling salesman problem," *Neurocomputing*, vol. 62, pp. 267–292, 2004.

[7] M. Dorigo and L. M. Gambardella, "Ant colonies for the travelling salesman problem," *Biosystems*, vol. 43, no. 2, pp. 73–81, 1997.

[8] X. H. Shi, Y. C. Liang, H. P. Lee, C. Lu, and Q. X. Wang, "Particle swarm optimization-based algorithms for TSP and generalized TSP," *Information Processing Letters*, vol. 103, no. 5, pp. 169–176, 2007.

[9] M. Padberg and G. Rinaldi, "A Branch-and-Cut Algorithm for the Resolution of Large-Scale Symmetric Traveling Salesman Problems," *SIAM Review*, vol. 33, no. 1, pp. 60–100, 1991.

[10] K. Helsgaun, "An effective implementation of the Lin–Kernighan traveling salesman heuristic," *European Journal of Operational Research*, vol. 126, pp. 106–130, oct 2000.

[11] G. Clarke and J. W. Wright, "Scheduling of Vehicles from a Central Depot to a Number of Delivery Points," *Operations Research*, vol. 12, no. 4, pp. 568–581, 1964.

[12] M. Gendreau, A. Hertz, and G. Laporte, "New Insertion and Post Optimization Procedures for the Traveling Salesman Problem," *Operations Research*, vol. 40, no. 6, pp. 1086–1095, 1992.

[13] Y.-t. Wang, J.-q. Li, K.-z. Gao, and Q.-k. Pan, "Memetic Algorithm based on Improved Inver–over operator and Lin–Kernighan local search for the Euclidean traveling salesman problem," *Computers & Mathematics with Applications*, vol. 62, pp. 2743–2754, oct 2011.

[14] G. C. Onwubolu, *New Optimization Techniques in Engineering*, ch. Optimizing, pp. 537–565. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004.

[15] P. Merz, *Memetic Algorithms for Combinatorial Optimization Problems: Fitness Landscapes and Effective Search Strategies*. PhD thesis, 2000.

[16] P. Moscato and C. Cotta, "A Modern Introduction to Memetic Algorithms," in *Handbook of Metaheuristics*, vol. 146, pp. 141–183, 2010.

[17] W. Hart, N. Krasnogor, and J. Smith, "Memetic evolutionary algorithms," *Recent advances in memetic algorithms*, pp. 3–27, 2005.

[18] NVIDIA, *NVIDIA CUDA C Programming Guide*. No. 350, 2014.

[19] R. L. Graham, T. S. Woodall, and J. M. Squyres, "Open MPI: A Flexible High Performance MPI," *Proceedings 6th Annual International Conference on Parallel Processing and Applied Mathematics*, pp. 228–239, 2005.

[20] J. A. Castro, M. A. Castro Liera, and I. Castro Liera, *Programación paralela aplicada en optimización*. La Paz, B.C.S.: Instituto Tecnológico de La Paz, 1 ed., 2012.

[21] M. A. Castro Liera, *Un algoritmo genético distribuido con aplicación en la identificación difusa de un proceso fermentativo*. Ph.d., Universidad Central, "Marta Abreu"de Las Villas, 2009.

[22] L. Davis, "Job Shop Scheduling with Genetic Algorithms," in *First International Conference on Genetic Algorithms*, (NJ, USA), pp. 136–140, L. Erlbaum Associates Inc., jul 1985.

[23] G. Reinelt, "TSPLIB–A Traveling Salesman Problem Library," *INFORMS Journal on Computing*, vol. 3, no. 4, pp. 376–384, 1991.

[24] Y. T. Wang, J. Q. Li, K. Z. Gao, and Q. K. Pan, "Memetic Algorithm based on Improved Inver-over operator and Lin-Kernighan local search for the Euclidean traveling salesman problem," in *Computers and Mathematics with Applications*, vol. 62, pp. 2743–2754, 2011.

ISBN: 978-607-97128-2-2