BENEMÉRITA UNIVERSIDAD AUTÓNOMA DE PUEBLA

FACULTAD DE CIENCIAS DE LA COMPUTACIÓN

“IMPLEMENTACIÓN PARALELA DE UN ALGORITMO GENÉTICO PARA RESOLVER EL PROBLEMA DEL AGENTE VIAJERO”

TESIS

PARA OBTENER EL TÍTULO DE:

INGENIERO EN CIENCIAS DE LA COMPUTACIÓN

PRESENTA

EDGAR MARTÍNEZ VARGAS

ASESORA

M.C. MARCELA RIVERA MARTÍNEZ

PUEBLA, PUEBLA. SEPTIEMBRE DE 2017

Dedicatoria

A mis padres, Rufina y Luis

Que siempre me apoyaron incondicionalmente
en la parte moral y económica para que
logrará terminar mi carrera profesional.

¡Gracias!

A mis hermanos, Luis, Victor, Rigo, Ángel, Jorge y Dani

Gracias por apoyarme en todo momento
y estar siempre conmigo.

A mi abuelita, Juanita †

Por su ejemplo de humildad y trabajo.

Agradecimientos

Le agradezco a Dios por ser mi fortaleza en los momentos de debilidad y por brindarme una vida llena de aprendizajes, experiencias y sobre todo felicidad.

Mi más sincero agradecimiento a mi asesora y guía de ésta tesis, MC. Marcela Rivera Martínez, gracias por haberme dado la oportunidad de trabajar con usted, por haber tenido la paciencia necesaria para ayudarme y por transmitirme su conocimiento.

También, quiero expresar un agradecimiento especial a la Dra. María de Lourdes Sandoval Solís y al MC. Luis René Marcial Castillo por su grata colaboración y por sus valiosos consejos en la realización de ésta tesis.

Gracias a mis amigos, Kevin A., Kike, Serna, Edith, Cely, Alexs y José Luis. Porque gracias a la convivencia con ellos, mi permanencia en la facultad de Ciencias de la Computación fue muy agradable. Gracias por compartir conmigo sus vidas y experiencias.

Finalmente, externar mi gratitud a cada uno de los profesores que compartieron sus conocimientos conmigo durante mi estancia en la universidad.

Índice general

Introducción **8**

Capítulo 1. Estado del arte **11**

Capítulo 2. Algoritmos genéticos **15**

Definición 15

Funcionamiento 16

Aplicación 17

Diagrama de un AG simple 17

Capítulo 3. Problema del agente viajero **28**

Historia 28

Definición 30

Formulación 31

Complejidad computacional 32

Capítulo 4. Implementación de algoritmo genético secuencial **35**

Representación 35

Función de aptitud 37

Generar la población 38

Evaluación de cada individuo 38

Selección de parejas 39

Reproducción de parejas 41

Mutación 42

Capítulo 5. Algoritmos genéticos paralelos **44**

Computación paralela 44

Algoritmo genético paralelo 48

Clasificación 48

Maestro-Esclavo 49

Grano fino 49

Grano grueso 50

Paradigmas de programación 51

Superlinealidad 51

Capítulo 6. Implementación de un algoritmo genético paralelo (OpenMP) 53

OpenMP 53

Algoritmo propuesto 58

Capítulo 7. Experimentos y resultados del algoritmo genético secuencial y el paralelo 59

Datos generales del nodo de la supercomputadora 60

Pruebas sobre el operador de Selección 60

Pruebas sobre el operador de Mutación 61

Comparación entre el algoritmo genético secuencial y paralelo 62

Speedup 63

Conclusiones 68

Referencias 70

Índice de figuras

Capítulo 2

1.Diagrama del funcionamiento de un algoritmo genético. 18

2. Representación de un cromosoma 19

3. Función de aptitud = 3 19

4. Selección por ruleta. 22

5. Selección por ranking 23

6. Cruzamiento basado en un punto 25

7. Cruzamiento basado en dos puntos. 25

8. Mutación basada en inversión. 26

9. Mutación basada en scrambled. 27

10. Mutación basada en intercambio. 27

11. Mutación basada en desplazamiento. 27

Capítulo 3

1. Representación de un ciclo hamiltoniano. 29

2. Juego inventado por Hamilton. 29

3. Representación gráfica del problema del agente viajero. 31

4. Clases de complejidad computacional 34

Capítulo 4

1. Representación numérica de los lugares a visitar. 36

2. Representación de un cromosoma o individuo 36

3. Grafo no dirigido de 5 ciudades. 37

4. Cromosoma inicial. 38

5. Evaluación de un individuo. 39

Capítulo 5

1. Memoria compartida 47

2. Memoria distribuida..... 47

3. Representación del esquema maestro-esclavo**.** 49

4. Representación del esquema de grano fino. 50

5. Representación del esquema de grano grueso..... 50

Capítulo 6

1. Modelo de procesamiento Fork-Join. 54

2. Tipos de planificación en un bucle paralelo. 57

Capítulo 7

1. Tiempos de ejecución para el problema de 1291 ciudades..... 65

2. Speedup para el problema de 1291 ciudades..... 66

Índice de tablas

Capítulo 4

1. Matriz de costos. 37

2. Selección de parejas basado en ruleta por costos. 40

Capítulo 7

1. Comparación de las técnicas de selección. 61

2. Comparación de las técnicas de mutación..... 62

3. Costos obtenidos del algoritmo genético secuencial, paralelo y los reportados en la literatura. 63

4. Tiempos de ejecución variando la cantidad de hilos. 64

Introducción

El problema del agente viajero o TSP por sus siglas en inglés (Traveling Salesman Problem) es ampliamente estudiado en el área de las Ciencias de la Computación, 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 solución en un tiempo de cómputo razonable. Entre mayor sea el tamaño del problema, en este caso, cuanto mayor sea la cantidad de ciudades mayor será el tiempo que tardará una computadora en calcular o encontrar la solución.

Si bien el origen del problema no se conoce con exactitud, algunas versiones remontan su origen en el año de 1832. Surgiendo en una guía de agentes viajeros de recorridos a través de Alemania y Suiza, sin embargo, en un inicio el problema carecía de un análisis matemático como lo conocemos hoy en día. La primera ocasión que fue estudiado por matemáticos fue durante los años de 1930 en Viena y Harvard, destacándose Karl Menger, quien definió el problema mediante el algoritmo de fuerza bruta. No obstante, fue hasta la década de 1950 que obtuvo un mayor impacto en los grupos científicos de Europa y Estados Unidos.

La solución al problema del agente viajero consiste en hallar el camino más corto entre *n* ciudades, tomando en consideración que cada ciudad solamente puede ser visitada una vez antes de llegar de nuevo al punto de partida. El concepto de camino más corto se puede aplicar para distintas situaciones, entendiendo que la solución siempre busca minimizar la distancia, el tiempo o el costo de realizar un traslado entre los nodos, que no necesariamente tienen que ser ciudades, sino que pueden ser puntos de soldadura, estaciones, entre otros.

Un campo de aplicación importante para este problema es en la industria logística para la transportación o reparto de productos, ya que se busca mejorar una ruta de entrega para seguir la más corta. Otra área de oportunidad es para el caso de turismo y agencias de

viajes; aun cuando los agentes de viajes no tienen un conocimiento explícito del problema del agente viajero, las compañías dedicadas a este giro utilizan un software que hace todo el trabajo. Finalmente, en el trabajo de la ciencia se utiliza en la robótica con el objetivo de resolver problemas de fabricación para minimizar el número de desplazamientos al realizar una serie de perforaciones en un circuito impreso.

En la actualidad, el continuo incremento 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 porque permiten, fácilmente, un diseño paralelizable. 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 [2] para paralelizarlo. Además, el propósito es reducir los tiempos de respuesta y encontrar soluciones buenas al aplicarlo a grandes problemas.

El trabajo se divide en 8 capítulos, el capítulo uno describe el estado del arte del problema, es decir, los avances y descubrimientos actuales que se tienen sobre el problema del agente viajero.

En el segundo capítulo se enuncia la definición de un algoritmo genético, así como su funcionamiento y aplicación. También se muestra y se explica cada uno los pasos del diagrama de flujo simple de éstos algoritmos.

En el capítulo 3 se describen de manera formal los antecedentes y la definición del problema del agente viajero, así como la forma en que se codifica o se representa este problema en un algoritmo genético.

El capítulo 4 presenta la primera y más básica implementación de un algoritmo genético. Es decir, la codificación en lenguaje C del algoritmo de forma secuencial, también

se muestran los experimentos y resultados obtenidos de los problemas citados en la literatura de los benchmark [3].

El capítulo 5 explica los modelos o topologías, maestro-esclavo, grano grueso y grano fino, para implementar un algoritmo genético con el objetivo de paralelizarlo. Además, los paradigmas de programación que se pueden aplicar a esas topologías, tales como OpenMP y MPI.

El capítulo 6 es similar al capítulo 4, ya que muestra la implementación paralela del algoritmo, el programa se codifica mediante la utilización de las directivas de OpenMP.

El capítulo 7 expone una comparativa entre los experimentos y resultados obtenidos con los dos tipos de implementaciones, es decir, tanto de la aplicación secuencial como de la versión paralela. Adicionalmente, las soluciones halladas se contrastan con los problemas benchmark mencionados anteriormente.

Por último, el capítulo 8 presenta las conclusiones sobre los capítulos anteriores y detalles como el rendimiento de cómputo.

Capítulo 1

Estado del arte

Una de las técnicas más sencillas de implementar para resolver el problema del agente viajero es la conocida como fuerza bruta [4], la cual consiste en evaluar todas las posibles permutaciones de rutas y, la gran ventaja es que ofrece la solución más óptima, es decir, el camino más corto. No obstante, este método, también presenta un gran inconveniente, ya que cuando el número de ciudades es elevado entonces el número de permutaciones posibles crece de manera exponencial, de modo que esta técnica se vuelve impracticable en la vida real puesto que se necesitarían de bastantes recursos computacionales.

Otro procedimiento para encontrar solución es el método del vecino más cercano [5], este consiste en que una vez que se haya seleccionado un punto de partida, el algoritmo evalúa y elige el nodo más cercano, después de que se ha trasladado al siguiente nodo más cercano, se marca al nodo anterior como visitado y se vuelve a realizar el mismo procedimiento, así hasta volver al punto de partida establecido. Este método no garantiza la solución óptima, como el caso de la técnica anterior, sin embargo, suele brindar buenas soluciones. Además, el tiempo de duración para ofrecer una solución es más eficiente en comparación con el método anterior.

Branch and bound (ramificación y poda) [6], es un método más que brinda soluciones a problemas de optimización, la meta de este algoritmo es encontrar el valor mínimo de una función. Básicamente, en un algoritmo de Ramificación y Poda se realizan tres etapas. La primera de ellas, denominada de “selección”, se encarga de extraer un nodo de entre el conjunto de los nodos vivos; un nodo vivo es un nodo con posibilidades de ser ramificado, es decir, que no ha sido podado. La forma de escogerlo va a depender directamente de la estrategia de búsqueda que se decida para el algoritmo. En la segunda etapa, la “ramificación”, se construyen los posibles nodos hijos del nodo seleccionado en el paso anterior. Por último se realiza la tercera etapa, la “poda”, en la que se eliminan algunos de

los nodos creados en la etapa anterior. Esto contribuye a disminuir en lo posible el espacio de búsqueda y así atenuar la complejidad de estos algoritmos basados en la exploración de un árbol de posibilidades. Aquellos nodos no podados pasan a formar parte del conjunto de nodos vivos, y se comienza de nuevo por el proceso de selección. El algoritmo finaliza cuando encuentra la solución, o bien cuando se agota el conjunto de nodos vivos. Este método presenta una mejor eficiencia en problemas de tamaño pequeño o mediano, ya que para problemas grandes sucede lo mismo que el primer método.

Las redes neuronales [7], han mostrado ser muy potentes en la solución de problemas que tienen como única manera de solución algoritmos aproximados, el problema del agente viajero es un tipo de ellos. El modelo de la red neuronal para este problema es la siguiente:

Se tiene dos capas de neuronas, una capa de entrada de dos neuronas para las ciudades, cada una de estas dos neuronas recibirá las coordenadas de posición de cada ciudad, es decir, la posición del nodo en el plano (coordenadas x, y), la segunda capa tendrá inicialmente varias neuronas. Cada neurona de la red neuronal, en la capa de competición tendrá asociado dos pesos un peso *wx* y uno *wy* que serán inicializados en forma aleatoria, luego la idea principal es que después de cierto estimulo enviado desde la capa de entrada con sus dos neuronas, este estimulo se propague por todas las neuronas de la capa de competición, y se encuentre una neurona ganadora, la cual se obtiene mediante una medida de distancia euclidiana entre la neurona ingresada y las demás neuronas de la capa de competición.

La Búsqueda Tabú [8] es un método metaheurístico que puede utilizarse para resolver problemas de optimización combinatoria, tales como el problema del agente viajero. La búsqueda tabú utiliza un procedimiento de búsqueda local o por vecindades para moverse iterativamente desde una solución x hacia una solución $x*$ en la vecindad de x, hasta satisfacer algún criterio de parada. Para poder explorar regiones del espacio de búsqueda que serían dejadas de lado por el procedimiento de búsqueda local, la búsqueda tabú modifica la estructura de vecinos para cada solución a medida que la búsqueda progresa. Las soluciones admitidas para $N^*(x)$, el nuevo vecindario, son determinadas mediante el uso

de estructuras de memoria. La búsqueda entonces progresa moviéndose iterativamente de una solución x hacia una solución $x*$ en $N^*(x)$.

Todos los métodos anteriores se consideran técnicas tradicionales o secuenciales, lo que conlleva a un gran inconveniente, por lo general, este tipo de algoritmos presentan la enorme desventaja ya que en medida que crece el tamaño o la instancia del problema el tiempo de respuesta para hallar la solución se vuelve compleja. Es decir, que se vuelve imposible poner en práctica estas técnicas para problemas verdaderos puesto que en situaciones reales las dimensiones de un problema son enormes por lo que una computadora necesitaría trabajar con grandes volúmenes de datos y, en la actualidad, un equipo de cómputo convencional no tiene la capacidad de almacenar y computar demasiados datos. Y, en el supuesto caso de que se cuente con tal equipamiento de computo, el tiempo en encontrar una solución incremente exponencialmente por lo que, inclusive, hallar una solución puede tardar cientos de años para un problema mayor.

Una alternativa a esta problemática son las técnicas de programación de alto rendimiento (performance), más comúnmente llamadas técnicas de procesamiento paralelo, estas constituyen un conjunto de métodos computacionales utilizados para la resolución de problemas sobre sistemas con una capacidad de procesamiento superior al modelo tradicional. 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.

Con respecto a lo antes mencionado, los algoritmos heurísticos se han venido utilizando para resolver problemas de optimización y búsqueda en los que no existen métodos eficientes para resolver todas sus instancias. Los métodos de computación

evolutiva y, en particular, algoritmos genéticos, se han destacado como métodos propicios, ya que su fácil codificación paralela permite resolver los problemas complejos, tal como es el problema del agente viajero.

Algunos artículos que proponen soluciones al problema del agente viajero mediante algoritmos genéticos paralelos son los siguientes: GRISLAS [9], un algoritmo genético paralelo que combina los modelos de grillas e islas para encontrar buenas soluciones; el modelo de Grillas (modelo celular) basa su trabajo sobre individuos distribuidos sobre una malla bidimensional, un individuo por cada celda. La interacción genética está restringida a los vecinos “cercanos” de cada individuo, mientras que el modelo de islas se basa en la estructura espacialmente distribuida de las poblaciones naturales. La población se divide en subpoblaciones disjuntas donde en cada una de ellas se ejecuta en forma paralela un algoritmo genético secuencial simple con intercambios regulares de individuos entre estas subpoblaciones según una cantidad de generaciones preestablecida y algunos parámetros determinados. Y el modelo PPREGA (Parallel Pattern Reduction Enhanced Genetic Algorithm) [10], es un método escalable para reducir el tiempo de cálculo de los algoritmos genéticos.

También ha sido enfrentado mediante el algoritmo de recocido simulado [11], 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. Una ventaja es que con el simple hecho de aplicar este método se puede reducir considerablemente el tiempo polinómico del problema.

Otra técnica heurística para resolver el problema es aplicando el algoritmo de colonia de hormigas [12], 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 el siguiente capítulo se presenta el tema de los algoritmos genéticos que se desarrollaron en este trabajo de tesis.

Capítulo 2

Algoritmos genéticos

En este capítulo se menciona una breve definición de los algoritmos genéticos, cómo es que surgieron, así como el funcionamiento de los mismos. Además, se explica a qué tipo de problemas se puede aplicar y cuál es la manera adecuada de representarlos y codificarlos. Por último, se presenta el diagrama general de un algoritmo genético simple y se describe cada uno de los pasos que lo conforman.

Definición

"Los algoritmos genéticos (AG) fueron introducidos por John Holland a finales de la década 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" [13].

Los algoritmos genéticos son métodos adaptativos que brindan soluciones a problemas de búsqueda y optimización. Básicamente, imitan el comportamiento o proceso evolutivo de los seres vivos. Es decir, con el transcurso de las generaciones, las poblaciones evolucionan en la naturaleza conforme a los principios de la selección natural y la supervivencia de los más fuertes, postulados por Darwin. Por tal motivo, este tipo de algoritmos se consideran aptos para desarrollar soluciones a problemas del mundo real. La evolución de dichas soluciones depende en buena medida de una adecuada codificación de las mismas.

Funcionamiento

El proceso evolutivo que durante años ha caracterizado a los seres vivos reside en que los individuos de la población compiten entre sí en la búsqueda u obtención de recursos, tales como alimento o un lugar donde vivir, inclusive rivalizan en la búsqueda de una pareja.

Solo aquellos individuos que tienen mayor capacidad para competir serán los individuos que tienen mayor porcentaje o probabilidad de sobrevivir, ya que son estos mismos quienes pueden propagar sus genes a través de descendientes a lo largo de las generaciones. Por el contrario, aquellos grupos de individuos débiles o menos aptos para sobrevivir, con el pasar del tiempo sus genes se irán descartando de la población lo que posiblemente conlleve a una extinción total.

De esta manera, cada individuo de la población representa una solución factible a un problema real dado. Cada individuo tiene asignado un valor que se calcula a través de una "función de aptitup". En la naturaleza esto equivaldría a la capacidad que posee un ser vivo para contender por determinados recursos. Cuanto mayor sea la adaptación de un individuo al problema, mayor será la probabilidad de que sea seleccionado para reproducirse, cruzando su componente genético con otro individuo seleccionado de igual forma. Este cruce producirá nuevos individuos descendientes, los cuales contienen algunas de las características de sus padres. En caso contrario, si la efectividad que tiene un individuo para adaptarse al problema es casi nula entonces tendrá menor probabilidad de que el individuo sea seleccionado para la reproducción.

Este acontecimiento genera una nueva población de posibles soluciones, ésta nueva población se denominan "hijos". La cual, en teoría, reemplaza a la población anterior por la razón de que ahora los nuevos individuos comparten las mejores características genéticas de la población anterior. De este modo, a lo largo de las generaciones las buenas características se propagan a través de la población, favoreciendo el cruce de los individuos mejor adaptados.

Aplicación

Como ya se ha mencionado en párrafos previos, la aplicación más común de los algoritmos genéticos ha sido en la solución de problemas de optimización, donde han manifestado ser muy eficientes y confiables. No obstante, no todos los problemas pudieran ser apropiados para resolverlos mediante esta heurística, para asegurarse de que es conveniente u oportuna esta técnica, antes de aplicarla se recomienda, en general, tomar en cuenta las siguientes características:

- **Definir una función de aptitud:** Cada individuo de la población debe tener asignada un valor que se calcula con una función dada.
- **Codificación:** Se refiere a la manera en que se representará cada individuo dentro de la población, existen distintas formas de codificar éstos, por ejemplo, mediante cadenas binarias (0 y 1), cadenas de números enteros, números reales, incluso a través de letras. Siendo las más populares las cadenas binarias.

Diagrama de un AG simple

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

A continuación se describe cada uno de los pasos que conforman el diagrama de la figura 1.

- **Representación de parámetros**: Los AG utilizan un vocabulario que es adoptado de la genética. Por ejemplo, los individuos que integran a la población también se les denomina “cromosomas”. Los individuos o

cromosomas están conformadas por unidades ordenadas linealmente llamados alelos, ver figura 2.

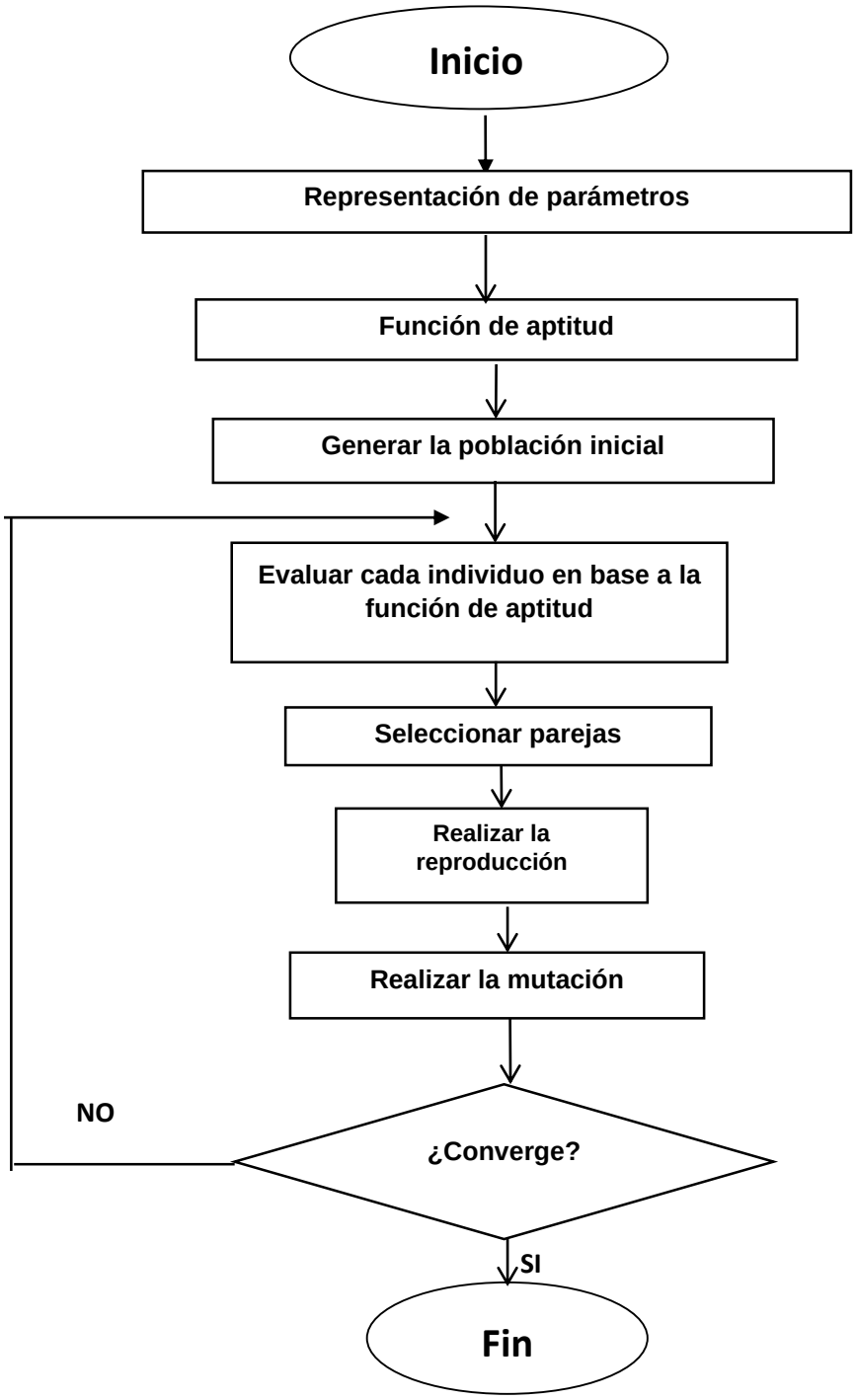


Figura 1.Diagrama del funcionamiento de un algoritmo genético.

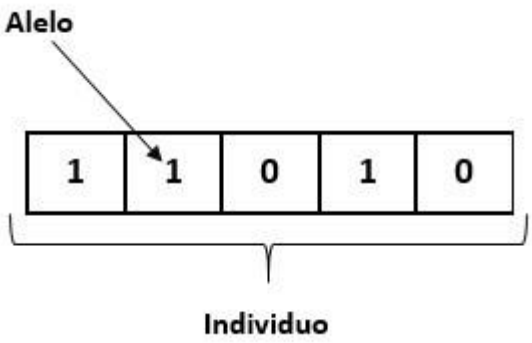


Figura 2. Representación de un cromosoma

La representación del cromosoma o individuo puede variar, esto depende del tipo de problema que se está resolviendo. Generalmente, se representa mediante cadenas que contiene valores binarios (0 y 1) como es caso de la figura anterior, a través de cadenas que contiene números ya sean reales o enteros, también se puede presentar el caso en la cual se conforma de letras o caracteres.

- **Función de aptitud**: Representa una función matemática o de evaluación que sirve para calcular el valor de cada individuo que refleja su aptitud o grado de efectividad frente a sus demás compañeros. Esta función ayudará a catalogar a los individuos más aptos de la población.

  Por ejemplo (ver figura 3), si la función de aptitud es una función que simplemente debe contabilizar el número de unos (1's) presentes en el individuo.

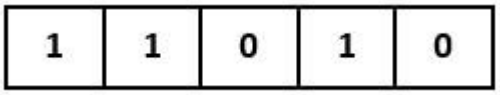


Figura 3. Función de aptitud = 3

- **Generar la población:** La población inicial se genera aleatoriamente, permitiendo toda la gama de posibles soluciones (el espacio de búsqueda). El número de individuos que conformarán la población deberá ser un parámetro de entrada del programa.

- **Evaluar la población:** Simplemente se aplica la función de aptitud a cada uno de los individuos de la población para conocer su valor de aptitud.

- **Seleccionar parejas:** Una vez que se haya realizado la evaluación total de la población, el siguiente paso es elegir los individuos que posteriormente intercambiarán sus genes para generar nuevos cromosomas. Dado que se pretende generar una población con mejor material genético, en teoría, se busca seleccionar a "los mejores individuos", es decir, aquellos que tengan un alto grado de efectividad con respecto a la función de aptitud.

  Los mecanismos de selección son variados, distinguiéndose tres grupos fundamentales:

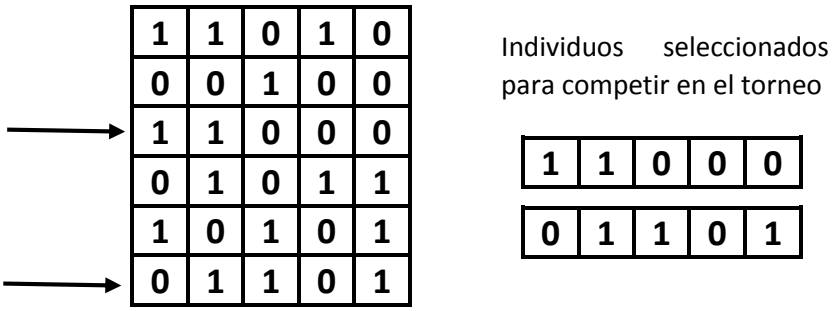
  - **Selección por torneo**

    Es un procedimiento de selección de padres en el cual la idea consiste en escoger al azar un número (*K)* de individuos de la población, tamaño del torneo, después seleccionar el mejor individuo de este grupo y repetir el proceso hasta que el número de individuos seleccionados sea igual al tamaño de la nueva población (*P),* llamada padres. Por lo general, el tamaño del torneo es de 2.

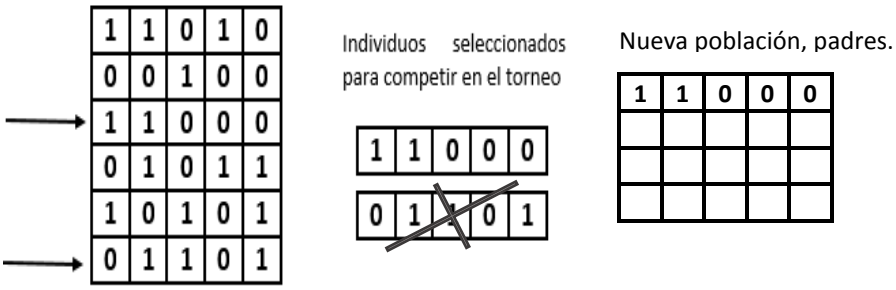
    Por ejemplo, si se tiene la siguiente población y si *K*=2, *P*=4

1	**1**	**0**	**1**	**0**
0	**0**	**1**	**0**	**0**
1	**1**	**0**	**0**	**0**
0	**1**	**0**	**1**	**1**
1	**0**	**1**	**0**	**1**
0	**1**	**1**	**0**	**1**

    Paso 1. Seleccionar 2 individuos aleatoriamente, ya que es el tamaño del torneo.



Paso 2. Elegir al mejor individuo de los dos seleccionados. Supóngase que el primer individuo es quien tiene mejor valor de aptitud, entonces se selecciona y se agrega a la nueva población (padres).



Paso 3. Regresar al paso 1 y repetir hasta completar la nueva población. Para este caso, se repite 3 veces más el procedimiento.

- **Selección basada en la ruleta**

  En esta técnica, los padres se seleccionan de acuerdo a su valor de aptitud. Los mejores individuos (con mejor aptitud) son los que cuentan con mayor probabilidad de ser elegidos. El proceso genera una ruleta en la cual cada uno de las porciones representa a un individuo. La porción de la ruleta que le toca a cada individuo es proporcional a su función de aptitud. Por tal razón, los mejores individuos se llevarán las mayores porciones y, al revés ocurrirá con los peores (ver figura 4).

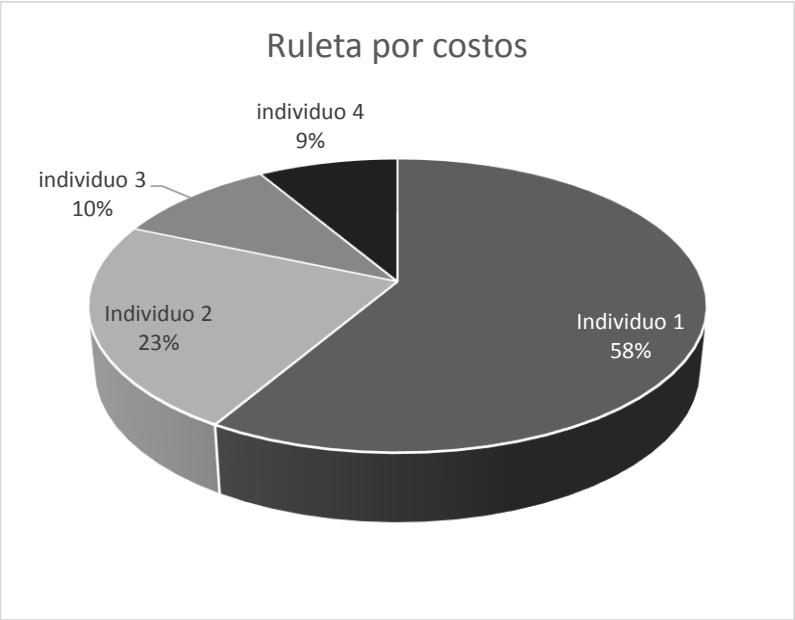


Figura 4. Selección por ruleta.

Resulta claro que los individuos con mejor función de aptitud son los que más a menudo son elegidos.

Existe un algoritmo para realizar este proceso:

1. **Suma total:** Calcular la suma total acumulada de los valores de aptitud de todos los individuos de la población.
2. **Elegir un número aleatorio r:** Generar un número aleatorio entre 0 y la Suma total.
3. **Seleccionar individuo:** Recorrer la población acumulando nuevamente los valores de aptitud. Cuando la suma que se lleve sea mayor o igual a *r* seleccionamos el individuo donde se vaya recorriendo.

- **Selección por ranking**

Esta técnica es muy similar a la de ruleta puesto que igualmente se representa a través de una ruleta.

La diferencia radica en que el método anterior presenta un problema, y es que si algún individuo o cromosoma ocupa el 80% de la ruleta,

claramente dejará con menor oportunidad de ser elegidos a los individuos menos aptos. La selección por ranking evita este inconveniente.

Ahora, los cromosomas son ordenados de acuerdo a su ranking de valor de aptitud. De tal forma que si se tienen *n* cromosomas el individuo con peor valor de aptitud se le asignará un 1 y el que tenga el mejor valor de aptitud se le asignará la *n*. Tomemos como ejemplo la figura de la técnica anterior y aplicando este método quedaría de la siguiente manera, ver figura 5.

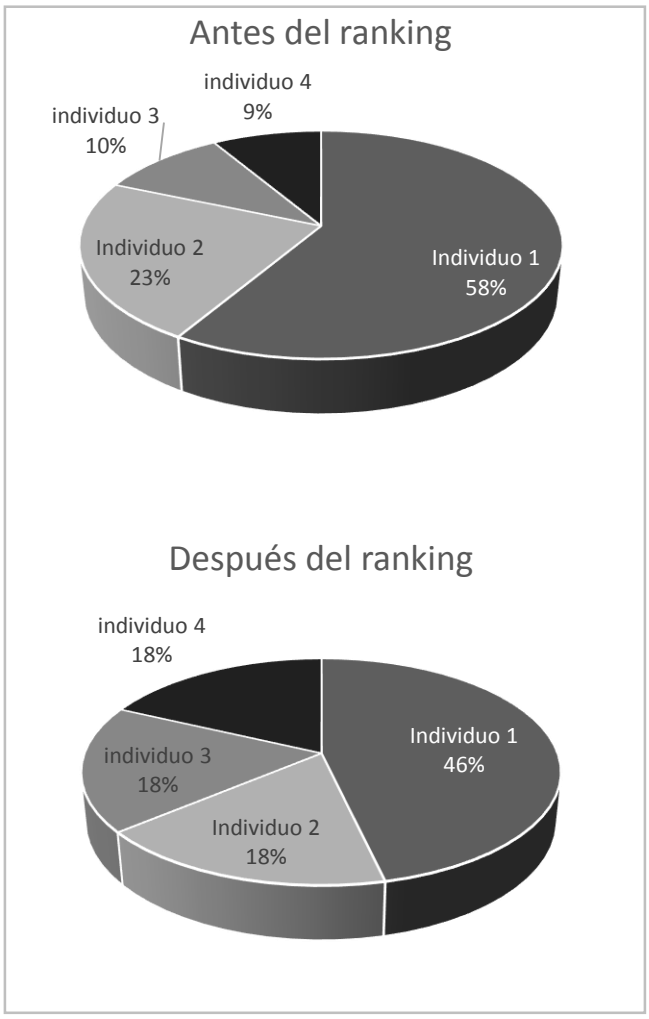


Figura 5. Selección por ranking

De este modo todos los individuos tienen mayor oportunidad de ser seleccionados. Una posible desventaja de esta práctica es que puede

hacer que el algoritmo converja lentamente a la solución, ya que los mejores individuos se diferencian apenas de los peores.

- **Cruzar parejas:**

  La cruza de los códigos genéticos de individuos exitosos favorece la aparición de nuevos individuos que heredan de sus ancestros características deseables. En el contexto de los algoritmos genéticos reproducirse significa que, dados dos individuos seleccionados en función de su grado de adaptación, éstos pasen a formar parte de la siguiente generación o, al menos, mezclen sus códigos genéticos para generar "hijos" que posean un código híbrido. Es decir, los códigos genéticos de los individuos se cruzan. Existen muchos mecanismos de cruzamiento y todos tienen por objeto que el código de un individuo A y el de uno B, previamente seleccionados, se mezclen, es decir, se fragmenten y recombinen para formar nuevos individuos con la esperanza de que éstos hereden de sus progenitores las características deseables.

  Este operador genético actúa como explorador, es decir, su función es descubrir áreas prometedoras en el espacio de búsqueda. Al igual que el operador anterior, en este también existen diferentes métodos que permiten realizar la reproducción o cruza entre los individuos previamente seleccionados. A continuación se mencionan los dos métodos más populares:

  - **Cruzamiento en un punto**

    La técnica consiste en tomar a dos individuos o cromosomas de la población "padres" y elegir aleatoriamente un punto que dividirá en dos segmentos a los cromosomas seleccionados. Se crean dos nuevos individuos, hijos. Para formar al primer hijo se toma el primer segmento del cromosoma 1, es decir, el padre 1, y se copia al hijo 1, seguidamente, se toma el segundo segmento del padre 2 y se agrega

al hijo 1. Para crear al segundo hijo el proceso es el mismo, solo que ahora se copia el primer segmento del padre 2 y el segundo segmento del padre 1, ver figura 6.

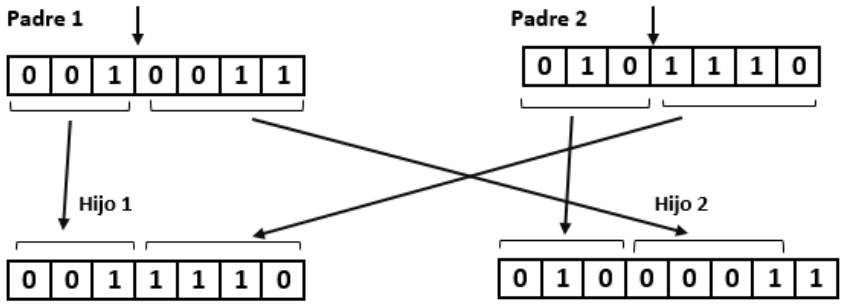


Figura 6. Cruzamiento basado en un punto

- **Cruzamiento en dos puntos (PMX)**

  El procedimiento es similar al anterior, ahora se seleccionan dos puntos al azar y se obtienen tres segmentos en cada padre. El procedimiento para generar al hijo 1 es el siguiente:

  Se selecciona el primer segmento del padre 1, después se copia el segundo segmento o segmento intermedio del segundo padre y posteriormente, se agrega el tercer segmento del padre 1.

  Para crear al hijo 2, se extraen los segmentos laterales, segmento 1 y 3, del segundo padre y se combinan en el centro el segmento intermedio del padre 1, ver figura 6.

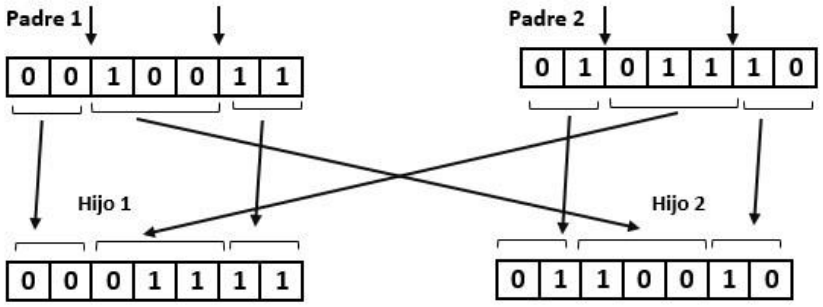


Figura 7. Cruzamiento basado en dos puntos.

- **Mutar:**

  Se aplica a cada hijo de manera individual, y consiste en la alteración aleatoria (normalmente con probabilidad pequeña) de cada gen componente del cromosoma. Si bien puede en principio pensarse que el operador de cruce es más importante que el operador de mutación, ya que proporciona una exploración rápida del espacio de búsqueda, éste último asegura que ningún punto del espacio de búsqueda tenga probabilidad cero de ser examinado, es decir, que éste operador explota la zona de búsqueda, además, es de suma importancia para asegurar la convergencia de los algoritmos genéticos.

  Enseguida se muestran algunos métodos de mutación [14]. Los cromosomas ahora están simbolizados con cadenas de números enteros, esto solo para variar un poco la representación de un individuo.

  - **Inversión**:

    Se generan dos números aleatorios entre 1 y el tamaño del cromosoma que dará como resultado una subcadena, posteriormente esta subcadena se invierte, ver figura 7.

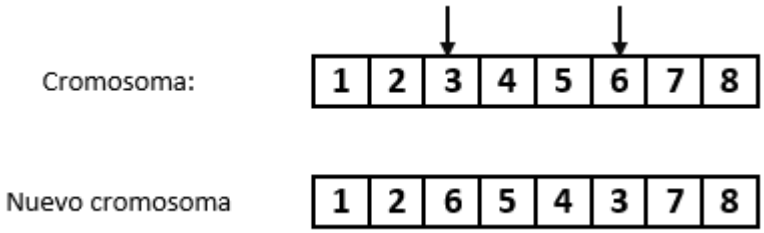


Figura 8. Mutación basada en inversión.

  - **Scrambled (Revuelto):**

    El procedimiento es similar al anterior puesto que se eligen dos números al azar comprendidos en el rango del cromosoma, los elementos o alelos que conforman la nueva subcadena se reordenan de manera aleatoria dentro de la misma, ver figura 8.

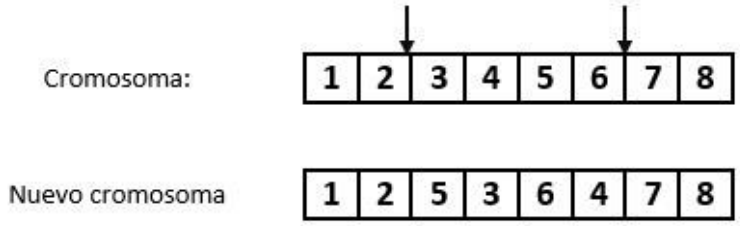


Figura 9. Mutación basada en scrambled.

- **Intercambio:**

  Esta técnica es más simple y consiste en seleccionar dos números aleatorios entre 1 y el tamaño del cromosoma, estos dos valores serán dos posiciones de la cadena del individuo. Posteriormente, se intercambia el contenido de dichas posiciones, ver figura 9.

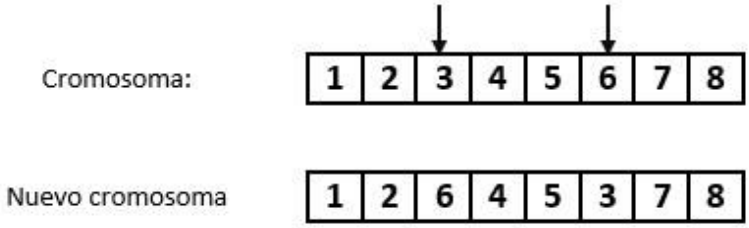


Figura 10. Mutación basada en intercambio.

- **Desplazamiento:**

  Nuevamente, se obtienen dos números aleatorios entre 1 y el tamaño del cromosoma, posteriormente, el valor contenido en la segunda posición se agrega antes de la primera posición y los demás elementos se recorren o desplazan a la derecha, ver figura 10.

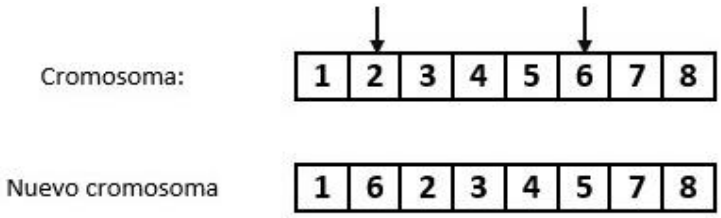


Figura 11. Mutación basada en desplazamiento.

Capítulo 3

Problema del agente viajero

Como ya se ha destacado en la parte introductoria, el problema del agente viajero o TSP, por sus siglas en inglés: Travelling Salesman Problem, es uno de los problemas de optimización más famosos y tiene una complejidad computacional NP-completo, un problema fácil de enunciar pero difícil de resolver. En este capítulo se describen los antecedentes del problema, cómo es que se originó toda esta problemática, se presenta la definición formal y su representación matemática del mismo y, finalmente, se explica el significado de complejidad computacional.

Historia

En 1832 se menciona el problema en un manual del agente viajeros con ejemplos de recorridos por Alemania y Suiza, pero sin un tratamiento matemático. De ahí el nombre de "problema del agente viajero" aunque su aplicación puede adaptarse a otros ejercicios como, por ejemplo, en el área de la robótica.

Problemas relacionados al agente viajero fueron estudiados por William Rowan Hamilton y Thomas Penyngton Kirkman. De ahí el nombre de la solución al problema que se le conoce como ciclo hamiltoniano; un ciclo hamiltoniano (ver figura 1), representado mediante un grafo, es una sucesión de aristas adyacentes que visita todos los vértices del grafo una sola vez y, además, el último vértice visitado es adyacente al primero.

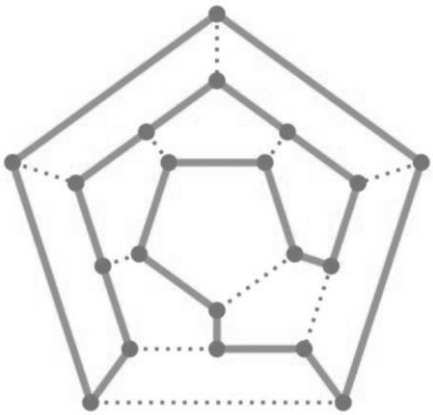


Figura 1. Representación de un ciclo hamiltoniano.

En el año de 1857, el mismo William R. Hamilton inventa un juego con un dodecaedro de madera que vendió a un fabricante de juguetes de Dublín, un dodecaedro es un poliedro que tiene doce caras en forma de un pentágono y que en total tiene 20 vértices. Cada uno de los vértices correspondía a 20 ciudades importantes del mundo. El objetivo era hallar una ruta o camino que uniera los 20 vértices sin pasar más de una vez por un mismo vértice, en pocas palabras, encontrar un ciclo hamiltoniano.

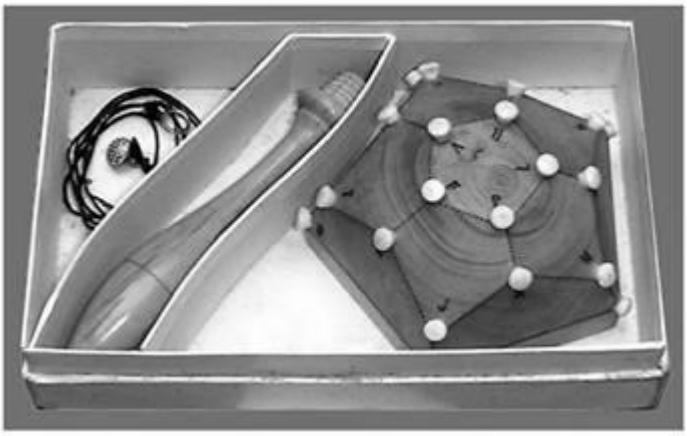


Figura 2. Juego inventado por Hamilton.

Merrill Flood fue el responsable de divulgar el nombre del TSP. John Williams incitó a Flood en 1948 a popularizar el TSP en la corporación RAND (Research ANd Development) en Santa Mónica, California, Estados Unidos, motivado por el propósito de crear talentos intelectuales. Otra razón de la popularidad del problema fue su íntima conexión con el

problema del transporte. Sin embargo, no fue hasta años posteriores, durante 1950 y 1960 que el problema incremento su divulgación entre la comunidad científica.

Richard M. Karp demostró en 1972 que el problema del ciclo de Hamilton era NP-completo [15]. Esto suministra una explicación matemática de la dificultad computacional aparente de encontrar rutas óptimas. Se hizo un gran avance en la década de 1970 y 1980, cuando Grotschel, Padberg, Rinaldi [16] y otros lograron resolver exactamente los casos con un máximo de 2,392 ciudades. En la década de 1990, Applegate, Bixby, Chvtal, y Cocine desarrollaron el programa Concorde [17] que se ha utilizado en muchas soluciones de registro recientes, logrando soluciones óptimas para instancias de 85,900 ciudades.

Gerhard Reinelt publicó el TSPLIB que contiene la descripción de una biblioteca para el problema del viajero que está destinada a proporcionar a los investigadores un amplio conjunto de problemas de prueba de diversas fuentes y propiedades en 1991, es decir, una colección de instancias de referencias de diferentes dificultades, que ha sido utilizado por muchos grupos de investigación para la comparación de resultados, como por ejemplo, en 2006, Cook y otros calculan un recorrido óptimo a través de una instancia de 85,900 ciudades dada por un problema de diseño microchip [18].

Definición

"Dado un conjunto de ciudades y el costo de los viajes entre cada par de ellos, se debe encontrar la forma más barata de visitar todas las ciudades y volver a su punto de partida" [19].

En la definición anterior, entiéndase por la palabra "costo" como una variable que puede cambiarse o sustituirse en el problema por otros conceptos, tales como, minimizar la "distancia" entre las ciudades o minimizar el "tiempo" que toma desplazarse entre ellas. Dependiendo del caso que se esté resolviendo, los conceptos pueden modificarse pero la idea para resolver el problema se mantiene.

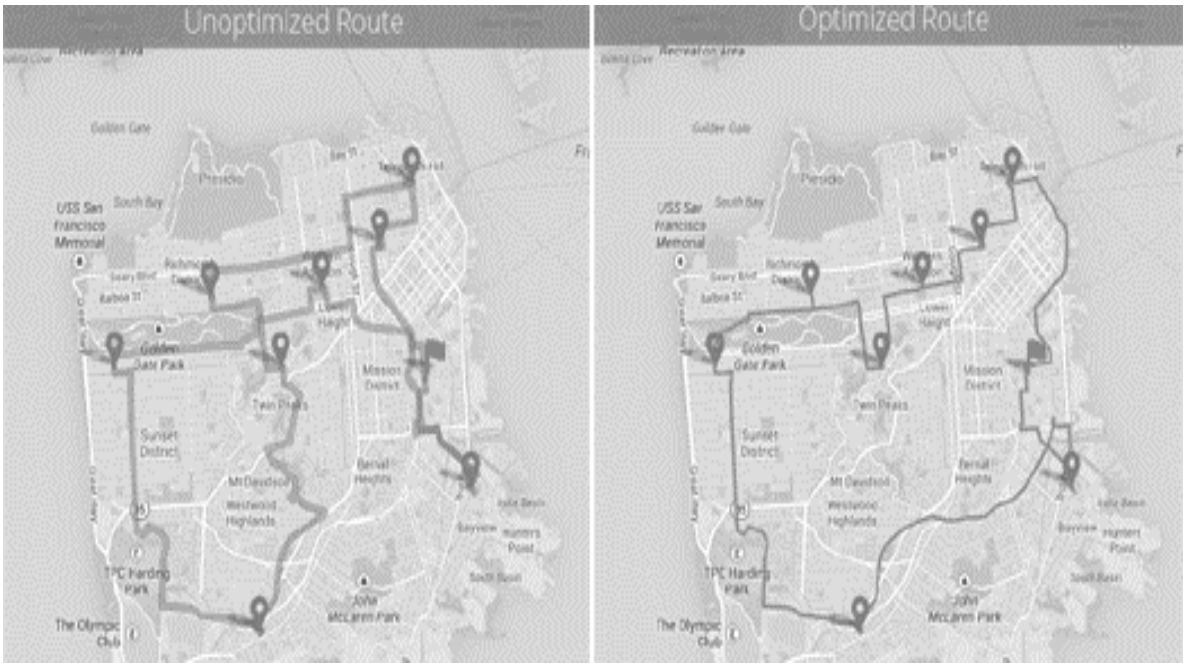


Figura 3. Representación gráfica del problema del agente viajero.

Formulación

El problema del agente viajero está dado de la siguiente forma: Sean *n* ciudades de un espacio geográfico, se especifica un punto o 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 binaria, es 1 en caso de que se visite la ciudad *j* después de visitar la ciudad *i* y 0 en caso contrario,

de esta manera se asegura que se cumpla la definición del problema, ya que limita a que el viajero visite solo una vez cada ciudad.

Retomando la técnica por fuerza bruta para resolver el problema; significa que hallar la solución más óptima equivale a evaluar todas las posibles permutaciones, es decir, analizar $n!$ permutaciones. Solo para un caso pequeño, por ejemplo, 10 ciudades se tendrían 10! permutaciones, eso quiere decir que existen 3 628 800 posibles caminos por evaluar, si el problema se incrementa al doble, esto es, 20 ciudades las posibles permutaciones no aumentan al doble sino que crecen exponencialmente, puesto que el factorial de 20 es igual a 2 432 902 008 176 640 000.

Como ya se mencionó anteriormente, la solución es una permutación de las n ciudades dadas, y se divide en dos tipos:

- Simétrico: En este caso, la matriz de costos C_{ij} es simétrica, es decir, el costo que genera viajar de la ciudad i a la ciudad j es el mismo que el de viajar de la ciudad j a la ciudad i.

- Asimétrico: En este caso, la matriz de costos Cij no es simétrica, es decir, el costo que se genera de viajar de la ciudad i a la ciudad j, en general, no es el mismo que el de viajar de la ciudad j a la ciudad i.

Complejidad computacional

Para comprender lo que es un problema NP es necesario abordar el tema de complejidad computacional, que estudia la "dificultad" de problemas de importancia teórica y práctica, como es el caso del problema del agente viajero.

El objetivo primordial que estudia la complejidad computacional es la de definir qué problemas son tratables y cuáles no, por ejemplo, los problemas de programación lineal se consideran tratables, dicho en otras palabras, son problemas que aunque la instancia del problema sea grande se asegura que se encontrará la solución óptima. No obstante, para el

problema que se está estudiando se considera intratable, ya que solo se puede hallar el valor óptimo para instancias pequeñas y de forma analítica.

Para conocer el orden de complejidad de un algoritmo se toman en cuenta dos recursos computacionales: El tiempo, número de pasos base de ejecución de un algoritmo para resolver un problema, y el espacio, cantidad de memoria utilizada para resolver un problema.

Existen distintas clases de complejidad computacional, entre las más conocidas se encuentran 3 y se clasifican en clase P, clase NP y clase NP-completo [20]:

- Clase P

  La clase de complejidad de los problemas que pueden ser resueltos en tiempo polinomial calculado a partir de la entrada por una máquina de Turing determinista es llamada P. Una máquina de Turing no es una máquina física sino un mecanismo lógico por medio del cual el cálculo puede descomponerse en iteraciones de operaciones concretas extremadamente simples (controladas por un "programa"). El principio de Turing es: "Cualquier procedimiento que puede ser descrito con precisión puede ser programado para que lo realice una computadora".

  Entonces los problemas pertenecen a la clase P si su solución puede ser descrita con precisión y ser programada para que la realice una computadora en tiempo que varíe polinomialmente (no exponencialmente, por ejemplo) en relación al tamaño de sus variables y/o parámetros de entrada. Estos problemas se denominan tratables.

  En pocas palabras, se dice que un problema pertenece a la clase P si existe un algoritmo que encuentra la solución óptima en un número de pasos acotado por un polinomio del tamaño del problema.

- Clase NP

  Es el conjunto de problemas que pueden ser resueltos en tiempo polinómico por una máquina de Turing no determinista. La importancia de esta clase de problemas de decisión es que contiene muchos problemas de búsqueda y de optimización para los que se desea saber si existe una cierta solución o si existe una mejor solución que las conocidas.

- Clase NP-Completo

  Un problema NP-completo es el sub-conjunto de los problemas de decisión en NP por esto todo problema en NP se puede transformar en cada uno de los problemas NP-completo. Se puede decir que los problemas NP-completo son los problemas con mayor dificultad de NP y es muy probable que no puedan formar parte de la clase de complejidad P.

  Un problema de decisión C es NP-completo si se cumple lo siguiente:

  - C es un problema NP.
  - Todo problema NP se puede transformar polinómicamente en C.

  Se puede demostrar que C es NP comprobando que una posible solución de C puede ser examinado en tiempo polinómico.

El problema del agente viajero es un problema NP-completo, sirve y ha servido como problema prueba (benchmark) para nuevos métodos de optimización. Es motivo de incontables investigaciones (artículos) en ciencias de la computación y en el área de investigación de operaciones.

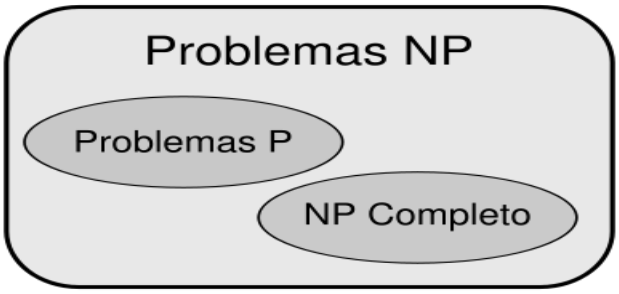


Figura 4. Clases de complejidad computacional

Capítulo 4

Implementación de algoritmo genético secuencial

En este capítulo se expone la manera en la que se codifica el problema del agente viajero en un algoritmo genético simple (el modelo es aquel que se muestra en la figura 1 del capítulo 2), es decir, cómo se representan los parámetros y de qué forma se calcula la función de aptitud en este problema, además, cómo se aplican los tres operadores genéticos fundamentales, los cuales son: selección de parejas, cruza o reproducción de parejas y mutación en un cierto porcentaje de la población, entre otros pasos necesarios. Dicha implementación, en primera instancia se programa de manera secuencial y se escribe bajo el lenguaje C.

Representación

Retomando algunas definiciones mencionadas en el capítulo 2, una de las maneras con mayor popularidad para representar un problema en un algoritmo genético es mediante cadenas binarias, es decir, a través de cadenas que contiene unos y ceros (1, 0) en cada una de sus posiciones. Sin embargo, en el contexto del problema del agente viajero no resulta conveniente aplicar una representación de ese tipo, lo más adecuado y eficaz es trabajar con cadenas de números enteros. De esta manera, simplemente bastará con enumerar las ciudades a visitar desde 1 hasta *n* ciudades, ver figura 1. Por lo tanto, la cadena (cromosoma o individuo) contendrá una posible permutación que va de 1 hasta *n* la cual simbolizará un posible recorrido. Un ejemplo, basándose en la figura 1 se representaría por medio de una cadena de tamaño de 7, la cual contendrá una posible permutación de 7 dígitos, ver figura 2. Cabe mencionar, que atendiendo a la definición del problema el cual menciona que el punto de partida también será el punto llegada del recorrido, se debe tomar en cuenta el costo que implica viajar del punto al que hace referencia la séptima posición al punto de la primera posición.

Figura 1. Representación numérica de los lugares a visitar.

5	**6**	**4**	**7**	**3**	**1**	**2**

Figura 2. Representación de un cromosoma o individuo

La figura 1 es una muestra visual de los puntos que se van visitar, esta representación también se puede exponer matemáticamente mediante un grafo. Por ejemplo, se tienen 5 ciudades simbolizadas por medio del siguiente grafo no dirigido (figura 3); un grafo no dirigido es un problema simétrico indicado en el capítulo anterior, es decir, el costo de viajar de la ciudad *i* a la ciudad *j* es el mismo que regresar de la ciudad *j* a la ciudad *i*. Las aristas del grafo significan los caminos validos que hay para desplazarse a otra ciudad y el valor que está sobre estas es el costo, distancia o tiempo asociado que hay de viajar de uno a otro destino.

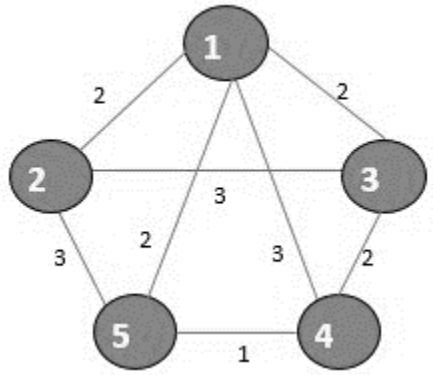


Figura 3. Grafo no dirigido de 5 ciudades.

La tabla 1 que aparece enseguida es una matriz la cual hace referencia a los costos asociados para trasladarse entre las distintas ciudades. Las “x” significan que no es posible llegar a ese destino estando en la ciudad actual, por ejemplo, no es posible viajar a la ciudad 4 estando en la ciudad 2.

Tabla 1. Matriz de costos.

x	2	2	3	2
2	x	3	x	3
2	3	x	2	x
3	x	2	x	1
2	3	x	1	x

Función de aptitud

La función de aptitud es la que está representada por la fórmula 1 que se muestra en el capítulo 3. Básicamente, lo que se calcula en esta expresión es la suma de los costos que hay para desplazarse entre ciudades que contiene cada individuo. Por lo tanto, para realizar este paso es necesario contar con la matriz de costos o distancias. En la implementación del programa, las distancias se calculan con la fórmula de la distancia euclidiana.

$$d = \sqrt{(X_2 - X_1)^2 + (Y_2 - Y_1)^2} \tag{2}$$

donde:

X_1 , Y_1 son las coordenadas de la ciudad actual y X_2 , Y_2 son las coordenadas de la ciudad destino.

Generar la población

La población inicial se genera creando individuos aleatoriamente, esto es, se forma una población de tamaño *x*; *x* es un parámetro de entrada ingresado por el usuario, con distintos individuos cada vez que se ejecuta el programa. El algoritmo para generar a un individuo aleatoriamente es el siguiente: Supóngase que el tamaño del problema consta de 5 ciudades, entonces el tamaño del individuo será una cadena o arreglo de 5 posiciones, ver figura 4.

1	**2**	**3**	**4**	**5**

Figura 4. Cromosoma inicial.

Algoritmo 1. Generar un nuevo individuo.

```
Para i:=1 hasta n; donde n es el tamaño del problema
      Generar un número aleatorio k entre 1 y n
      Intercambiar valor de individuo[i] con individuo[k]
```

Evaluación de cada individuo

Simplemente se calcula la función de aptitud para cada individuo de la población. A continuación, se explica mediante un problema de pocas ciudades con el objetivo de hacerlo entendible.

Se considera nuevamente la figura 3 y su respectiva tabla de costos, tabla 1, y supóngase que se tiene el siguiente individuo **[2, 3, 1, 4, 5]**. Por lo tanto, primero se toma el costo que hay de viajar de la ciudad 2 a la ciudad 3, después se suma el costo de la ciudad

3 a la ciudad 1, posteriormente, de la ciudad 1 a la 4 y así sucesivamente hasta visitar todas las ciudades que contiene dicho individuo , ver figura 5.

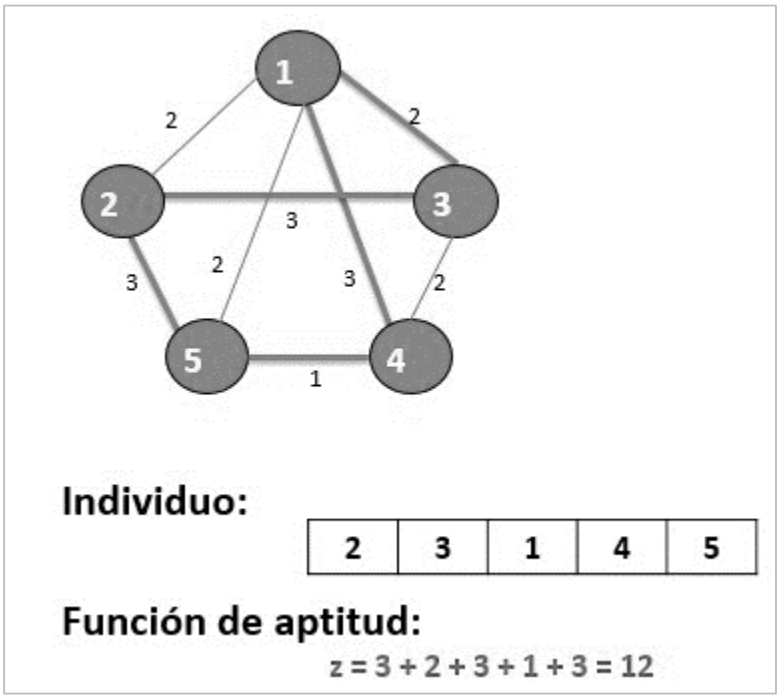


Figura 5. Evaluación de un individuo.

Nótese que de la ciudad 5 se suma el costo para volver a la ciudad de inicio, es decir, el costo de viajar de la ciudad 5 a la ciudad 2, esto es por la definición del problema que dice que el punto de partida también es el punto de finalización del recorrido.

Selección de parejas

Como se mencionó en el capítulo 2, existen mecanismos de selección muy variados. 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. Para calcular la fracción proporcional se necesita la siguiente ecuación:

$$p_i = \frac{f_i}{\sum_{j=1}^{x} f_j} \tag{3}$$

donde:

f_i es el valor de aptitud del individuo *i,* Σf_j es la sumatoria de los valores de aptitud de todos los individuos de la población. *X* es el tamaño de la población.

Por ejemplo, supóngase que se tiene una población de 5 individuos con su valor de aptitud y fracción proporcional, respectivamente (ver tabla 2). f_j será igual a 40. La última columna es la sumatoria de todas las fracciones proporcionales (Acumulación de proporcion) de cada individuo, está siempre debe dar como resultado la unidad (1).

Tabla 2. Selección de parejas basado en ruleta por costos.

Individuo	Valor de aptitud (f_i)	Fracción proporcional (p_i)	$\sum_{i=1}^{x} p_i$ = Sp_i
1	12	0.3	0.3
2	5	0.125	0.425
3	8	0.2	0.625
4	10	0.25	0.875
5	5	0.125	1

Posteriormente, se generan *Nbuenos* números de modo aleatorio entre 0 y 1 (*Nbuenos* se calcula dividiendo entre 2 al tamaño de la población total). En este caso, la mitad de 5 es 2.5; no obstante, el valor debe ser un número par, por lo cual *Nbuenos* se queda en 2. El siguiente algoritmo sirve para seleccionar a los *Nbuenos* individuos:

Algoritmo 2. Seleccionar *Nbuenos* individuos basado en ruleta por costos.

```
Para i:=1 hasta Nbuenos
      Generar un número aleatorio r
      j=1 y bandera = 0
      Mientras bandera == 0
            Si Sp_j < r
                  j++
            Sino
                  Seleccionar individuo
                  bandera:=1
```

Supóngase que los dos valores son: **0.153, 0.702.** Entonces los individuos seleccionados serían el 1 y 4.

Reproducción de parejas

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), 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. Enseguida se describe el algoritmo del PMX:

Algoritmo 3. Reproducción de parejas basado en PMX.

```
1. Se toman 2 individuos previamente seleccionados, ahora se
   denominan padres.
2. Se generan 2 números aleatorios entre 1 y el tamaño del cromosoma,
   estos serán los puntos de cruza de los padres.
3. Para crear el hijo 1, se copia el segmento intermedio del padre
   2, para formar el hijo 2 se copia el segmento intermedio del
   padre1.
4. Para completar los segmentos laterales de cada hijo se consigue
   mapeando los segmentos laterales de cada padre.
      Si el valor a copiar ya existe en el segmento intermedio del
      hijo, entonces
            Se sustituye por el valor que contenga esa misma posición
            pero en el segmento del otro padre.
      Sino
            Se mantiene el mismo valor que contiene ese padre.
```

Ejemplo:

Puntos de cruza

Padre1: [9, 8, 4 **|** 5, 6, 7 **|** 1, 3, 2, 10]

Padre2: [8, 7, 1 **|** 2, 3, 10 **|** 9, 5, 4, 6]

Los hijos quedarían en una primera instancia:

Hijo1: [X, X, X **|** 2, 3, 10 **|** X, X, X, X]

Hijo2: [X, X, X **|** 5, 6, 7 **|** X, X, X, X]

Para completar al Hijo1 e Hijo2, se copian primero los valores que no están en el segmento intercambiado.

Hijo1: [9, 8, 4 **|** 2, 3, 10**|** 1, X, X, X]

Hijo2: [8, X, 1 **|** 5, 6, 7 **|** 9, X, 4, X]

Por último, se mapean los valores que faltan en cada hijo. Por ejemplo, en la octava posición del hijo 1 el valor a copiar debería ser el número 3, sin embargo, este digito ya existe en el cromosoma. Por lo tanto, el valor sustituto es el número 6, ya que este número es el que se encuentra en la misma posición (del otro padre) donde ya se encuentra registrado el 3. El mismo procedimiento se realiza para completar los cromosomas. El resultado final de los dos nuevos individuos es el siguiente:

Hijo1: [9, 8, 4 **|** 2, 3, 10**|** 1, 6, 5, 7]

Hijo2: [8, 10, 1**|** 5, 6, 7 **|** 9, 2, 4, 3]

Mutación

En esta implementación, es decir, la versión secuencial, la técnica de mutación que se utiliza es el método por inversión. Busca cambiar aleatoriamente la información genética, el procedimiento es similar al operador genético anterior, en ésta se generan dos números diferentes aleatorios entre 1 y el tamaño del cromosoma, esta subcadena se invierte posteriormente.

Ejemplo:

Puntos de cruza

Individuo: [9, 8 **|** 4, 2, 3, 10**|** 1, 6, 5, 7]

Nuevo_ Individuo: [9, 8 **|** 10, 3, 2, 4 **|** 1, 6, 5, 7]

En un inicio se implementó la técnica de mutación sobre la población de los hijos, posteriormente, se aplicó sobre un determinado porcentaje de la población en curso. Dicho porcentaje es un parámetro de entrada que el usuario es capaz de ingresar en tiempo de ejecución. Se optó por implementar la mutación sobre la población en curso puesto que brinda mejores resultados.

Existen otros pasos más que complementan el modelo simple de un algoritmo genético, estos no son más que la actualización de la población. Es decir, primeramente calcular el valor de aptitud de la nueva población, denominada hijos, así como de los individuos que hayan sufrido algún cambio debido a la aplicación de la técnica de mutación. Posteriormente, unir las poblaciones tanto como la población en curso (padres e individuos mutados) y la nueva población, hijos. Una vez que se hayan combinado, se procede a ordenar de menor a mayor a todos los individuos para poder recortar a la población a su tamaño original, con esto se garantiza que los individuos mejor dotados se mantengan en la parte superior de la población y, al mismo tiempo, se descarten aquellos individuos que tengan un valor de aptitud con poca probabilidad de sobrevivir.

Capítulo 5

Algoritmos genéticos paralelos

La computación paralela se ha convertido en una parte fundamental en todas las áreas de cálculo científico, ya que permite la mejora del rendimiento simplemente con la utilización de un mayor número de procesadores, memorias y la inclusión de elementos de comunicación que permitan a los procesadores trabajar conjuntamente para resolver un determinado problema. En este capítulo, a grandes rasgos, se aborda el paradigma de programación paralela aplicada en los algoritmos genéticos; se presentan tres distintas formas que existen para hacer un programa en paralelo, a su vez, estas clasificaciones se pueden implementar con diferentes métodos de programación, tales como: paralelismo en datos y programación por paso de mensajes.

Computación paralela

Con la necesidad de mejorar o encontrar nuevos métodos para hallar solución a problemas de instancias grandes surge la computación paralela, este modelo conlleva a un nuevo sistema de programación, distinto del modelo tradicional o secuencial. Este método, básicamente, está basado en la ejecución de instrucciones de manera simultánea, lo cual permite ejecutar más instrucciones en un menor tiempo. Su funcionamiento es muy similar al método de "divide y vencerás" puesto que su propósito es dividir el tamaño original del problema en partes más pequeñas y computarlas de forma paralela.

El surgimiento del paralelismo está ligado a diferentes factores [21], tales como:

- Potencia expresiva de los modelos de procesamiento paralelo:
  Muchos problemas son más fáciles de modelar usando paradigmas paralelos, ya sea por la estructura que se usa para su resolución o porque el problema es intrínsecamente paralelo. Es decir, si desde el principio se puede pensar

en los mecanismos paralelos/concurrentes para resolver un problema, eso puede facilitar la implantación del modelo computacional. Esto podría permitir obtener mejores soluciones para los problemas a resolver, en tiempos razonables de ejecución. Así, este enfoque permite el surgimiento de modelos de cálculos diferentes, a los modelos secuenciales. En pocas palabras, además del procesamiento de paralelismo para mejorar los tiempos de ejecución se agrega la ganancia conceptual, al poder resolver los problemas con nuevos métodos de resolución hasta ahora imposibles de ejecutarse en máquinas secuenciales.

- Una mejor relación costo/rendimiento:

  En el orden económico, es muy importante tener máquinas con excelente relación costo/rendimiento. Normalmente, el mayor poder de cálculo genera una explosión de costos que a veces lo hacen prohibitivo. Una manera para lograr mayor poder de cálculo sin costos excesivos es hacer cooperar muchos elementos de cálculo de bajo poder, y por consiguiente, de bajos costos.

- La necesidad de mayor potencia de cálculo:

  Independientemente de que la potencia de los procesadores aumente, siempre habrá un límite que dependerá de la tecnología del momento. Para aumentar la potencia de cálculo, además de los progresos tecnológicos que permitan aumentar la velocidad de cálculo, se requieren nuevos paradigmas basados en cálculo paralelo. Es decir, una manera de aumentar la velocidad de cálculo es usar múltiples procesadores juntos. Así, un problema es dividido en partes, cada una de las cuales es ejecutada en paralelo en diferentes procesadores. La programación para esta forma de cálculo es conocida como programación paralela, y las plataformas computacionales donde pueden ser ejecutadas estas

aplicaciones son los sistemas paralelos y distribuidos. La idea que se usa es que al tener *n* computadores se debería tener *n* veces más poder de cálculo, lo que conllevaría a que el problema pudiera resolverse en *1/n* veces del tiempo requerido por el secuencial. Por supuesto, esto bajo condiciones ideales que raramente se consiguen en la práctica. Sin embargo, mejoras sustanciales pueden ser alcanzadas dependiendo del problema y la cantidad de paralelismo presente en el mismo.

La computación paralela se ha convertido en el paradigma dominante en la arquitectura de computadores, sin embargo, recientemente, el consumo de energía de los ordenadores paralelos se ha convertido en una preocupación.

Refiriéndose al sistema de los multiprocesadores, estos pueden contener una única memoria al que todos los procesadores son capaces de acceder, la cual se le denomina memoria compartida, o existe otro caso en el que cada procesador pude poseer su propia memoria, es decir, memoria distribuida. A continuación, se describen más a detalle los dos tipos de clasificación de memoria, así mismo, se ilustran con su correspondiente figura para conocer su arquitectura.

- Memoria compartida:

  Cada procesador tiene acceso a toda la memoria, cuentan con un espacio de direccionamiento compartido y la comunicación entre procesadores se realiza a través de la memoria. Para garantizar el intercambio de datos se necesitan primitivas de sincronismo. Las ventajas de este sistema son: fácil de programar, la velocidad para compartir datos entre procesadores es mayor en comparación con la memoria distribuida y tiene un bajo costo. Sus desventajas son las siguientes: el acceso simultáneo a la memoria puede convertirse en un problema importante, la escalabilidad que presenta al aumentar el número de procesadores es mínima, debido a que se puede generar un cuello de botella al intentar acceder a memoria demasiados procesadores.

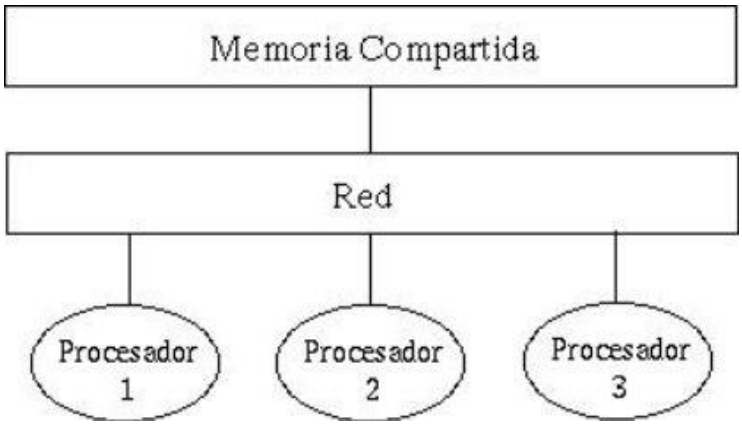


Figura 1. Memoria compartida

- Memoria distribuida:

  Este tipo de sistemas cuentan con su propia memoria local, la comunicación entre los procesadores únicamente se realiza compartiendo mensajes, es decir, si un procesador necesita conocer información sobre otro procesador entonces deberá enviar un mensaje solicitando dichos datos, a esta comunicación, generalmente, se le denomina "paso de mensajes". Su principal ventaja es: la escalabilidad, a diferencia de un sistema de memoria compartida, un sistema de memoria compartida es fácil de escalar, debido a que se puede agregar más memoria y procesadores a medida que la demanda de los recursos crece. Sus principales desventajas son: el acceso remoto a memoria es lento y el grado de complejidad de programación es complicada.

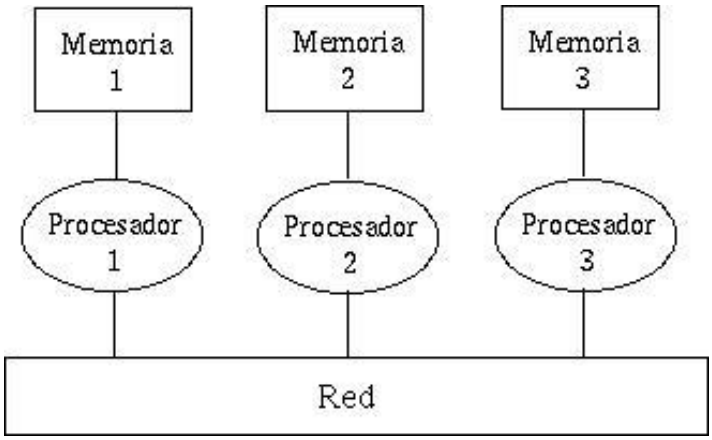


Figura 2. Memoria distribuida

A su vez, estas dos implementaciones se pueden incorporar en un solo sistema, logrando un sistema híbrido designado memoria compartida distribuida.

Algoritmo genético paralelo

Una enorme ventaja que presenta un algoritmo genético es que su estructura se adapta correctamente a la paralelización. Basta con observar la evolución natural, en sí es un proceso paralelo debido a que evoluciona utilizando varios individuos. Los principales métodos de paralelización de un AG consisten en la división de la población en varias subpoblaciones.

Clasificación

Existen diversas maneras de paralelizar un algoritmo genético. Una de las formas más conocidas consiste, básicamente, en paralelizar la evaluación y algunos operadores genéticos de los individuos manteniendo una única población, esta técnica se le denomina maestro-esclavo. Otra manera de paralelización general, consiste en hacer una ejecución de distintos AG secuenciales simultáneamente, ya sea que se implemente el primer o segundo método, la configuración del AG (figura 1 que se muestra en el capítulo 2) no presenta modificaciones. Los demás mecanismos si presentan alteraciones en la estructura del algoritmo, ya que dividen la población en subpoblaciones que evolucionan por separado e intercambian individuos cada cierto número de generaciones, a esta acción se le nombra operador de migración de individuos. Cuando las poblaciones son pocas y grandes, se tiene la paralelización de grano grueso. Ahora, si el número de poblaciones es grande y con pocos individuos en cada población entonces se tiene la paralelización de grano fino. Por último, existen algoritmos que combinan estas propiedades y que se denominan híbridos.

Maestro-Esclavo

Estos algoritmos trabajan con una única población de individuos que será gestionada por el nodo (procesador) maestro. La evaluación de la función de los individuos y/o la aplicación de los operadores genéticos puede ser realizada por los nodos esclavos. Su funcionamiento es relativamente simple, a cada uno de los nodos esclavos se le asigna una porción de la población total de individuos, estos realizan las acciones anteriormente mencionadas y devuelven el/los resultados al nodo maestro para continuar con la ejecución del programa. La comunicación entre maestro y esclavo puede hacerse de manera síncrona o asíncrona, es decir, el nodo maestro debe esperar a que todos los nodos esclavos finalicen sus operaciones correspondientes para poder continuar con el proceso o la otra forma es que el nodo maestro no espera a que cada nodo esclavo finalice su trabajo para continuar con la ejecución de las instrucciones. Sin embargo, con el último método se debe tener cuidado al momento de acceder a los recursos de memoria.

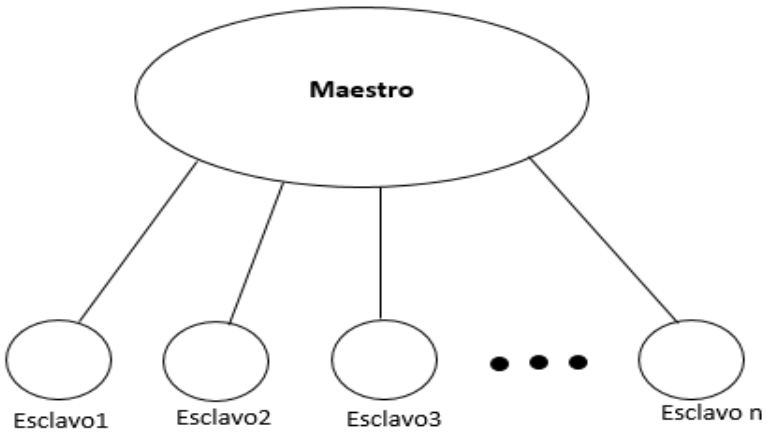


Figura 3. Representación del esquema maestro-esclavo**.**

Grano fino

También se les conoce como grid o parrilla debido a la distribución que disponen las poblaciones. 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 harán entre individuos que pertenezcan a un mismo vecindario; un vecindario está formado por un conjunto de individuos adyacentes.

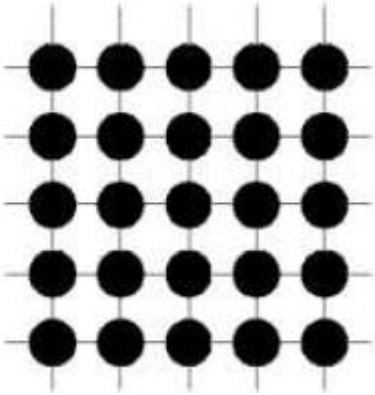


Figura 4. Representación del esquema de grano fino.

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 evoluciona independientemente, el porcentaje de migración será muy importante de cara a obtener resultados satisfactorios.

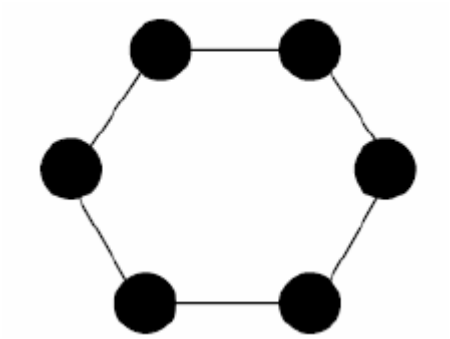


Figura 5. Representación del esquema de grano grueso.

Hay dos aspectos a tomarse en cuenta para la implementación de esta última clasificación, la primera es el número de individuos a intercambiar durante el operador de migración y el segundo es la frecuencia con la cual se intercambiarán los individuos.

Paradigmas de programación

Estas configuraciones se pueden implementar mediante tres distintos estilos de programación [22]:

- 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.

- 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).

- 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.

Superlinealidad

El enfoque principal de esta investigación es el análisis de los beneficios de una implementación paralela de los algoritmos genéticos frente al modelo tradicional, es decir, comparar el rendimiento de un algoritmo paralelo en relación a la programación secuencial.

Un beneficio primordial de los algoritmos genéticos paralelos (AGPs) es el tiempo de ejecución del programa, con respecto a este tema se introduce un nuevo concepto denominado “speedup” [23] el cual se refiere al cociente entre el tiempo de cómputo de un algoritmo genético secuencial y su versión paralela. A continuación se presenta la ecuación para calcular el factor de mejora de rendimiento o speedup:

$$S(n) = \frac{\mathrm{T}(1)}{T(n)} \qquad \textbf{(4)}$$

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.

En teoría, cuanto mayor sea el número de procesadores mayor será la rentabilidad de un programa paralelo, no obstante, esto no siempre se cumple en la práctica debido a que la comunicación necesaria entre los distintos procesadores también consume un tiempo, adicionando a esto, la porción del programa que no se pueda paralelizar. En relación a estas limitantes en la mejora del rendimiento, existe una manera de calcular la correspondencia entre la aceleración esperada de la implementación paralela de un algoritmo y la implementación serial del mismo algoritmo, esto se logra mediante la ley de Amdahl [24]:

$$A = \frac{1}{(1-P) + \frac{P}{N}} \qquad \textbf{(5)}$$

donde:

N es el número de procesadores y *P* es la porción paralelizable del programa [0,1], por ejemplo, si la ejecución de un programa secuencial tarda 20 segundos y al paralelizarlo y ejecutarlo en dos procesadores dura 10 segundos, entonces el speedup será de 50% y su porción paralelizable es de 0.5.

Capítulo 6

Implementación de un algoritmo genético paralelo (OpenMP)

Este capítulo es breve, únicamente se explica algoritmo genético paralelo que se propone. El código se escribe en el lenguaje C y se utiliza el paradigma de programación de paralelismo en datos, es decir, se ejecuta un solo proceso y el paralelismo se lleva a cabo lanzando distintos hilos durante la ejecución del programa, para esto se usan las directivas de OpenMP. Este modelo está basado en el diagrama de flujo (figura 1) mostrado en el capítulo 2. Se implementa con la configuración de maestro-esclavo, se trabaja con una sola población, la cual será gestionada por el hilo maestro, su función principal es particionar a la población y pasarla a los hilos esclavo para que estos realicen las operaciones necesarias de manera simultánea.

OpenMP

OpenMP [2] es una interfaz de programación (API) desarrollada para facilitar la programación paralela en sistemas de memoria compartida con arquitectura SMP (Symmetric Multi-Processor). En un inicio fue adoptada como un estándar informal en el año de 1997 y funcionaba únicamente para el lenguaje de Fortran, hoy en día, las nuevas versiones permitan adicionar las directivas de OpenMP a los programas secuenciales escritos en lenguajes de C/C++. Este paradigma usa explícitamente paralelismo multi-hilos, en programación, un hilo es un procedimiento que corre de forma independiente del programa principal. La idea central de este paradigma es desarrollar un programa secuencial que realice la tarea y brinde la solución, posteriormente, implementar las directivas de OpenMP para paralelizarlo. La principal ventaja de este estilo de programación es que el compilador es quien se encarga de la distribución de los datos guiado por un conjunto de directivas que introduce el programador.

OpenMP está basado en un modelo de procesamiento denomidado “Fork-Join”, el cual consiste en la ejecución del programa por un único hilo como en un programa secuencial, a este hilo se le nombra maestro; este ejecutará el programa hasta que encuentre una región paralela, una región paralela se reconoce en el código fuente por la inclusión de una directiva de OpenMP. Entonces, en esta parte se genera el fork; se trata simplemente de que el hilo maestro crea un grupo de hilos, hilos esclavos, los cuales ejecutarán de forma simultánea el conjunto de instrucciones contenidas en la región paralela. Por último, el hilo maestro recoge los resultados obtenidos de las operaciones realizadas por los hilos esclavos y a este momento se le llama join. El hilo maestro continúa con la ejecución del programa hasta encontrar nuevamente una región paralela, ver figura 1.

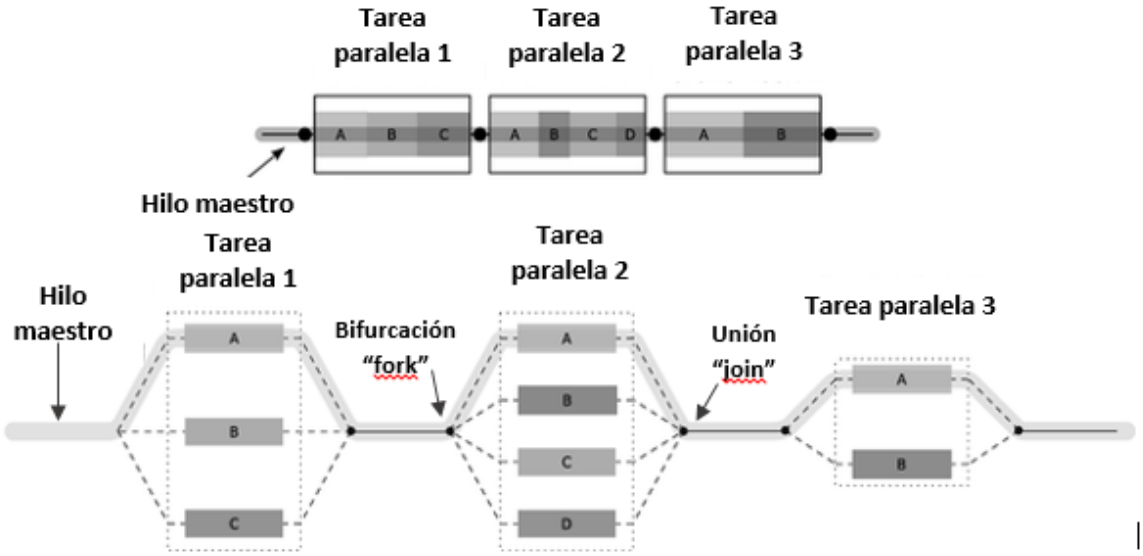


Figura 1. Modelo de procesamiento Fork-Join.

Para convertir una parte del programa en una región paralela se debe incluir dentro del código fuente una seria de directivas que encierran en un bloque parte del código y que el compilador de OpenMP será capaz de interpretar. La sintaxis de una directiva o pragma (también denominado constructor paralelo) escrita en lenguaje C es la que se muestra a continuación [25]:

#pragma omp parallel nombre-directiva [cláusulas] {

Bloque de instrucciones

}

El parámetro denominado *nombre-directiva* puede referirse a 3 tipos de regiones paralelas:

- Sections:

  Se refiere a secciones del programa que se ejecutan en paralelo.
- Bucle for:

  El ciclo es dividido para que cada hilo esclavo realice cálculos por separado.
- Single:

  El código es ejecutado por un único hilo. En este caso, se omite el parámetro *parallel.*

A su vez, cada región paralela puede hacer uso de algunas *cláusulas,* las cuales son de gran ayuda, su función es primordial debido a que permiten controlar el acceso a las variables contenidas en un bloque paralelo. A continuación, se mencionan las *cláusulas* más conocidas o de mayor utilidad, estas también se les denominan como cláusulas de alcance de datos y son aplicadas especialmente al momento de paralelizar el bucle for:

- Private(*Lista de variables*):

  Sirve para introducir una *lista de variables* que se consideran *privadas,* es decir, cada hilo o proceso almacenará una copia de esa lista de variables, esas variables no se inicializan con algún valor. Al finalizar el bloque paralelo, cada hilo elimina la copia de la variable que había guardado.
- Firstprivate(*Lista de variables*):

  Similar al anterior, cada hilo contiene su propia copia. La diferencia es que las variables se inicializan con el valor que contiene la variable antes de entrar a la región paralela.
- Lastprivate(*Lista de variables*):

  Al finalizar el bloque paralelo, cada variable almacena el valor obtenido en la última iteración del bucle o la última sección paralela.

- Reduction(*operador:variable):*

  Al final del bucle paralelo, todos los hilos compartirán sus resultados de una misma *variable* para actualizar la copia global del hilo maestro.
- Shared(*Lista de variables*):

  La *lista de variables* introducidas será del tipo compartida entre los distintos hilos creados.
- Default(*private|shared*):

  Indica en automático, cómo serán todas las variables. Ya sea que todas las variables sean compartidas o privadas.
- No wait:

  Si se especifica, los hilos no se sincronizan al final del bucle paralelo.
- Schedule(*tipo, chunk):*

  Decide cómo se distribuyen las iteraciones de un bucle entre los hilos, la planificación por defecto depende de la implementación. Existen tres distintas formas de realizar la distribución de iteraciones (ver figura 2):

  - Schedule(*static*,*chunk*):

    Distribuye un subconjunto de iteraciones en cada hilo de modo circular. El tamaño viene dado por "*chunk*", un parámetro introducido por el usuario/programador. Si no se especifica *chunk*, la distribución es por bloques.
  - Schedule(*dynamic*,*chunk*):

    El valor por defecto de *chunk* es 1, es casi igual que *static*, pero los bloques se reparten dinámicamente a medida que los hilos van finalizando.
  - Schedule(*guided*,*chunk*):

    Similar a *dynamic*, pero el tamaño del *chunk* comienza grande y cada vez disminuye para tener un mejor control de carga entre iteraciones. El parámetro *chunk* es opcional y especifica el tamaño mínimo que

deben utilizarse. De forma predeterminada, el tamaño de *chunk* es aproximadamente *N/p,* donde *N* es el número de procesadores y *p* es la cantidad de hilos.

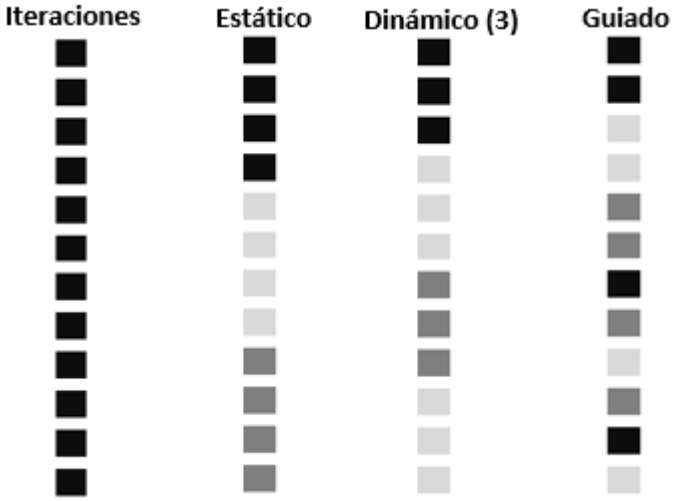


Figura 2. Tipos de planificación en un bucle paralelo.

Cuando aparece otro constructor paralelo dentro de una región paralela se le conoce como anidamiento paralelo. Siendo el caso, cada hilo esclavo crearía un nuevo grupo de hilos.

Existen otras directivas que son de gran utilidad al momento de sincronizar la ejecución de los distintos hilos, por ejemplo:

- Master:

  Solo el hilo maestro es quien ejecuta esta región, los demás hilos se saltan el bloque.
- Critical:

  Declaración de una región critica, a la cual todos los hilos deben ejecutar uno a la vez.
- Atomic:

  Asegura que una locación de memoria específica, va a ser actualizada de forma atómica, es decir, no se van a producir múltiples escrituras desde diferentes hilos.

- Barrier:

  Implementa una barrera en la que cada uno de los hilos deberá esperar al otro, una vez que todos hayan llegado a la barrera, continúan su trabajo en paralelo.
- Ordered:

  Se utiliza dentro de un bucle paralelo y su función es ejecutar ciertos bloques de manera secuencial.

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 terminan 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.

A continuación, se explican los segmentos (operadores) del algoritmo que se paralelizan, para esto se retoma el diagrama de flujo de un algoritmo genético mostrado en la figura 1 correspondiente al capítulo 2.

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.

Algoritmo 1. Algoritmo genético paralelo propuesto.

```
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
```

Capítulo 7

Experimentos y resultados del algoritmo genético secuencial y el paralelo.

En este capítulo se exponen los experimentos realizados sobre las distintas implementaciones para resolver el problema del agente viajero, así como los resultados obtenidos en cada una de ellas. El algoritmo propuesto se validó utilizando problemas prueba reportados en la literatura [3], en la supercomputadora del Laboratorio Nacional de Supercómputo del Sureste de México [26], usando el lenguaje de programación C [27]. Los problemas utilizados para las pruebas son TSP_small10, TSP_small20, data_100, tsplib_problem_a280, data_500, rat783, pr1002 y tsplib_problem_d1291; el tamaño del problema es 10, 20, 100, 280, 500, 783, 1002 y 1291 ciudades, respectivamente.

Se implementan distintas versiones del algoritmo genético (AG), variando el mecanismo de operador genético. Por ejemplo, en el operador de selección de individuos se implementa mediante ruleta por costo y por torneo, para el caso del operador de mutación se codifica a través de las técnicas de inversión, revuelto (scrambled), intercambio y desplazamiento, finalmente, el operador de reproducción se hace con la técnica de PMX o cruza de dos puntos. El objetivo de realizar distintas implementaciones es analizar los resultados obtenidos y escoger el mecanismo que mejor se adapte al problema.

Datos generales del nodo de la supercomputadora

24 procesadores Intel(R) Xeon(R) CPU E5-2680 v3 @ 2.50GHz, arquitectura x86_64, capacidad en memoria RAM 125 GB, versión de kernel Linux 2.6.32-504.8.1.el6.x86_64.

Pruebas sobre el operador de Selección

El primer experimento realizado fue la implementación del algoritmo genético variando el mecanismo de selección, el primero corresponde a la técnica de selección por ruleta de costos y el segundo método compete a la selección por torneo.

La tabla 1 muestra una comparación de los resultados obtenidos al ejecutar el programa con su correspondiente problema. El objetivo de esta prueba no es la de

encontrar los resultados que se reportan en los benchmark sino la de observar qué técnica de selección ofrece mejores soluciones; para esto, se ejecuta 5 veces cada versión del programa para los ya mencionados problema y se escoge el mejor resultado de cada versión. Por ejemplo, para el problema de 100 ciudades, se ejecuta 5 veces el programa mediante la técnica de selección de ruleta por costos y de esos 5 resultados obtenidos se elige el mejor, posteriormente, se ejecuta 5 veces el programa ahora con la técnica de selección de individuos por torneo y, nuevamente, se escoge la solución de menor valor.

Tabla 1. Comparación de las técnicas de selección.

Problema	Técnica	Resultado
10	ruleta	1324.55
	torneo	1324.55
20	ruleta	1944.47
	torneo	2034.16
100	ruleta	3973.77
	torneo	4180.73
280	ruleta	3357.12
	torneo	4648.18
500	ruleta	9637.47
	torneo	11213.99
783	ruleta	54716.82
	torneo	58807.09
1002	ruleta	360894.68
	torneo	361559.56
1291	ruleta	153371.06
	torneo	157497.04

Se puede observar de la tabla anterior que la técnica de selección de ruleta por costo ofrece mejores soluciones al problema. Solamente para el problema de 10 ciudades las dos técnicas alcanzan el mismo resultado.

Pruebas sobre el operador de Mutación

Un segundo experimento realizado es la codificación del algoritmo con 4 versiones o métodos de mutación distintas, las técnicas utilizadas son: inversión, revuelto, intercambio

y desplazamiento. Para cada problema prueba se realizaron 10 ejecuciones del algoritmo con distintos parámetros sobre la cantidad de generaciones, tamaño de la población y la cantidad de hilos. La técnica de selección utilizada es ruleta por costo y el porcentaje de mutación es de 0.3. La tabla 2 representa la comparación sobre los resultados obtenidos con cada uno de los mecanismos de mutación implementados. La última columna se refiere a los mejores resultados obtenidos hasta el momento.

Se nota, en la tabla 2, que se destacan dos técnicas por encontrar buenos resultados, la primera es inversión y la segunda es desplazamiento. Para los problemas de 10, 100 y 500 ciudades, con las 4 técnicas se alcanza la solución reportada en la literatura.

Comparación entre el algoritmo genético secuencial y paralelo

Un aspecto muy importante consiste en analizar los resultados finales que se obtienen con cada versión del algoritmo, es decir, tanto secuencial como paralelo y hacer una comparación con los problemas benchmark. El tipo de selección y mutación que se utilizaron son ruleta por costo y la técnica de inversión, respectivamente.

La tabla 3 reporta los costos menores obtenidos con las ejecuciones en el algoritmo genético secuencial, columna 3, y el algoritmo genético paralelo, columna 4, así como el mejor reportado en la literatura en la actualidad, última columna.

De la tabla 3, se hallaron mejores resultados, enmarcados en negrita, con el AG paralelo en comparación con el AG secuencial. Además, cabe resaltar que para el problema de 500 ciudades se alcanzó una solución menor a la reportada en la literatura.

Tabla 2. Comparación de las técnicas de mutación.

Ciudades	Técnica	Resultado	Benchmark
10	Inversión	1324.55	**1324.55**
	Scrambled	1324.55	
	Intercambio	1324.55	
	Desplazamiento	1324.55	
20	Inversión	1836.92	**1836.92**
	Scrambled	2002.15	
	Intercambio	1947.03	
	Desplazamiento	1836.92	

100	Inversión	3746.40	**3746.40**
	Scrambled	3746.40	
	Intercambio	3746.40	
	Desplazamiento	3746.40	
280	Inversión	2812.68	**2579**
	Scrambled	3239.13	
	Intercambio	3162.54	
	Desplazamiento	2831.71	
500	Inversión	8456	**8456**
	Scrambled	8456	
	Intercambio	8456	
	Desplazamiento	8448.00	
783	Inversión	12432.10	**8806**
	Scrambled	29228.07	
	Intercambio	25527.66	
	Desplazamiento	15064.91	
1002	Inversión	299334.65	**259045**
	Scrambled	907799.5	
	Intercambio	814653.43	
	Desplazamiento	312346.93	
1291	Inversión	140602.39	**50801**
	Scrambled	148724.10	
	Intercambio	150231.06	
	Desplazamiento	137225.34	

Tabla 3. 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	2821.76	**2579**
data_500	500	8862	**8448**	**8456**
rat783	783	29078.83	12432.10	**8806**
pr1002	1002	414509.87	299334.65	**259045**
tsplib_problem_d1291	1291	153672	84961.39	**50801**

Speedup

Una gran ventaja de paralelizar un programa, aparte de obtener mejores resultados, se presenta al disminuir el tiempo de ejecución. A continuación, la tabla 4 expone el tiempo que tarda en ejecutarse el programa del algoritmo genético tanto secuencial como paralelo. Además, se incluye un parámetro muy importante en un programa paralelo, es decir, la cantidad de hilos de ejecución. En este caso, se ejecuta con 2, 10, 24, 50 y 100 hilos, los

parámetros ingresados durante la ejecución del programa son los siguientes: para los problemas de 10 y 20 ciudades (generación=100, población=100 y porcentaje de mutación=0.3) y para los problemas restantes (generación=500, población=500 y mutación=0.3). La segunda columna representa la cantidad de hilos, con relación a la celda que muestra "Secuencial" significa que es la ejecución del algoritmo genético secuencial, para las celdas restantes se refiere al algoritmo genético paralelo ejecutado con diferentes cantidades de hilos.

Tabla 4. Tiempos de ejecución variando la cantidad de hilos.

Problema	Hilos	Tiempo
TSP_small10	Secuencial	0.0123
	2	0.0258
	10	0.0346
	24	0.04
	50	0.22
	100	0.416
TSP_small20	Secuencial	0.0257
	2	0.0349
	10	0.0405
	24	0.0423
	50	0.2119
	100	0.39
data_100	Secuencial	2.57
	2	2.06
	10	1.94
	24	3.2
	50	4.69
	100	5.72
tsplib_problem_a280	Secuencial	13.95
	2	9.82
	10	4.7
	24	4.31
	50	9.26
	100	10.31
data_500	Secuencial	38.24
	2	22.32
	10	10

	24	8.02
	50	14.32
	100	15.84
rat_783	Secuencial	114.83
	2	65.85
	10	21.54
	24	18.66
	50	20.9
	100	21.44
pr_1002	Secuencial	139.59
	2	83.76
	10	28
	24	22.8
	50	26.64
	100	28.33
tsplib_problem_d1291	Secuencial	249.48
	2	124.85
	10	50.08
	24	31.86
	50	36.76
	100	38.83

Con respecto a los resultados conseguidos sobre los tiempos de ejecución, se aprecia que para los dos primeros problemas, la duración es menor con el AG secuencial debido a que el AG paralelo también necesita un periodo para sincronizar el trabajo entre los distintos hilos. Para el caso de 100 ciudades, puede percibirse que el AG paralelo comienza a obtener mejoras, en relación al tiempo, y es suficiente ejecutarlo con 10 hilos. Para el resto de los problemas, se logra un menor tiempo al ejecutar el programa con 24 hilos.

A continuación, se presenta la figura 1 para poder visualizar y comparar de una mejor manera los tiempos de ejecución, la imagen muestra los tiempos correspondientes al problema tsplib_problem_d1291.

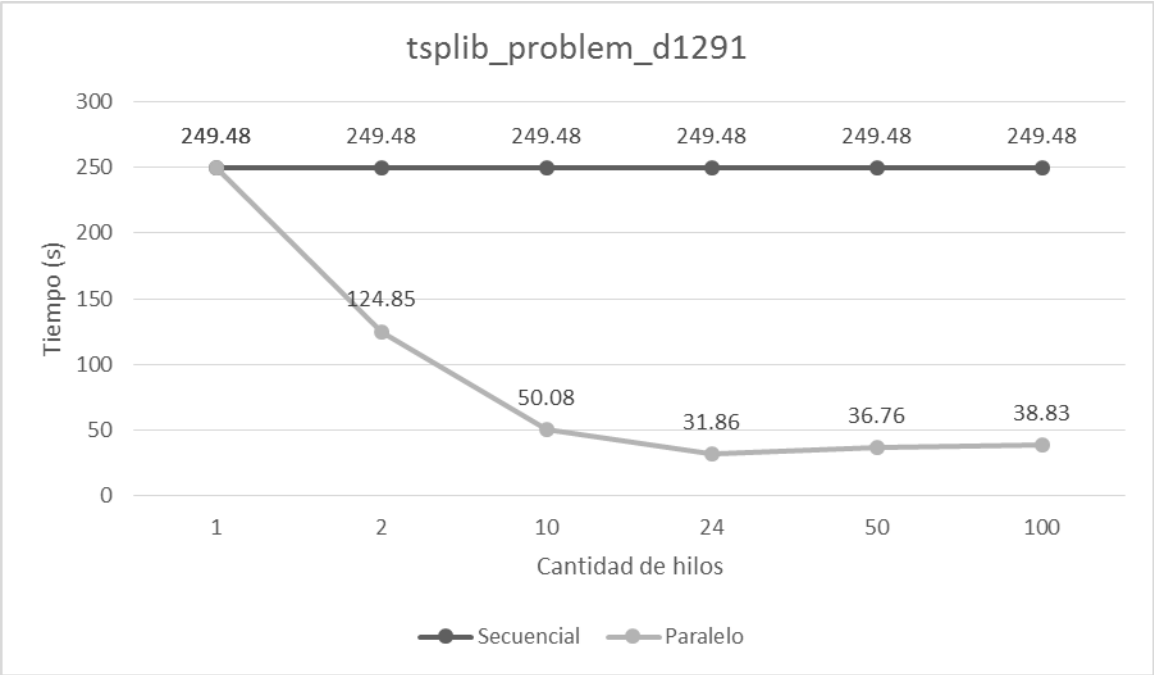


Figura 1. Tiempos de ejecución para el problema de 1291 ciudades.

Claramente se nota que la línea que representa el tiempo del algoritmo genético secuencial se encuentra por encima del algoritmo genético paralelo, solo al ejecutar el programa paralelo con dos hilos, el tiempo se reduce a la mitad; al correrlo con 10 hilos el tiempo vuelve a reducir, si bien, el tiempo conseguido no es proporcional al número de hilos, es decir, si con dos hilos el tiempo se redujo a la mitad entonces con 10 hilos se debería reducir diez veces resultando un lapso de 24.9 segundos, sin embargo, esto tiene sentido debido a lo que con anterioridad se ha mencionado, también se consume un tiempo sincronizando el trabajo entre los hilos.

Ahora, para calcular el speedup del programa se hace uso de la ecuación 4 del capítulo 5, para mostrar esta mejora de rendimiento se aplica nuevamente para el caso de 1291 ciudades. Enseguida, la figura 2 expone el factor de mejora del algoritmo paralelo.

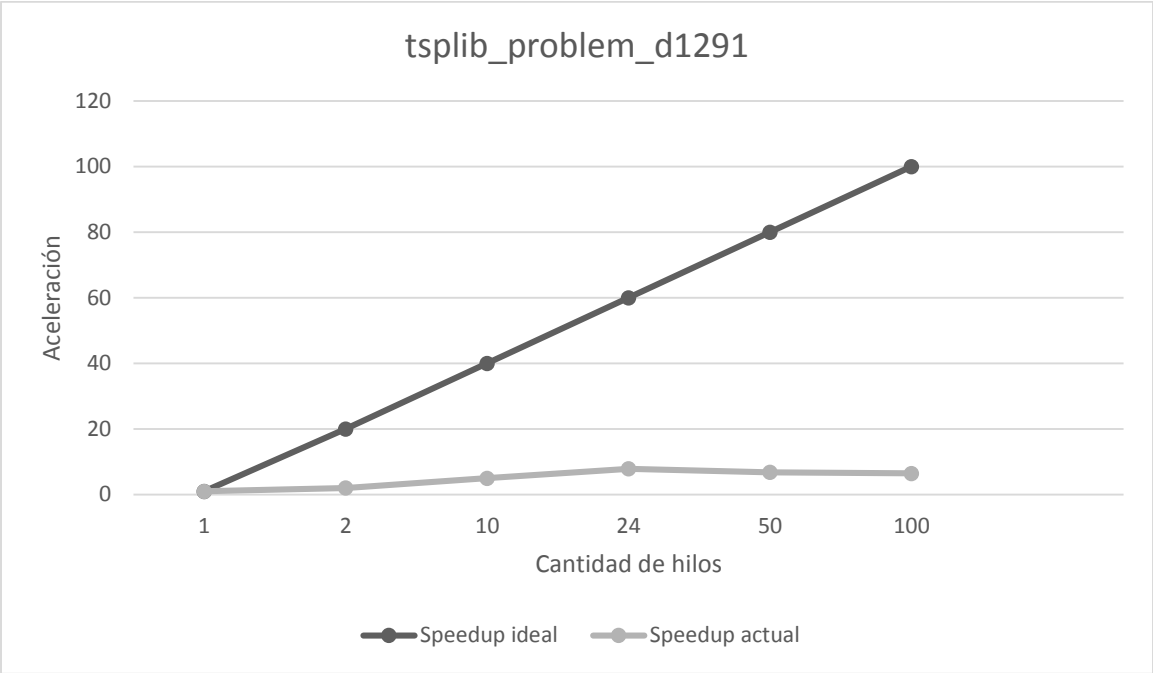


Figura 2. Speedup para el problema de 1291 ciudades.

La línea del speedup ideal o speedup lineal expresa el valor máximo que puede alcanzar, el valor máximo del speedup en teoría es igual a la cantidad de procesadores, por ejemplo, si el algoritmo dura un segundo con 1 procesador entonces con 5 procesadores no debería tardar más de 5 segundos. Entonces, en relación a la figura 2, se logra un speedup superlineal [28], es decir, el programa paralelo es mucho más rápido que su correspondiente secuencial.

Capítulo 8

Conclusiones

Los algoritmos genéticos tradicionales son de gran ayuda para problemas de búsqueda y optimización, sin embargo, estos algoritmos se vuelven insuficientes para problemas grandes, y su tiempo de ejecución se vuelve prohibitivo. La gran ventaja que presentan es su configuración que fácilmente se puede paralelizar y así poder aprovechar o tener un mejor rendimiento sobre el equipo de cómputo.

Con respecto al primer experimento realizado, observar tabla 1, se concluye que la técnica de selección de ruleta por costo brinda mejores resultados en comparación con el método implementado de selección por torneo. Para la gran mayoría de problemas se alcanzaron buenas soluciones con el mecanismo de ruleta por costo, excepto en el primer problema, 10 ciudades, en la cual las dos versiones lograron encontrar la mejor solución reportada en la literatura hasta este momento.

Observando los resultados registrados en la tabla 2, se concluye que de las 4 distintas técnicas de mutación que se implementaron, solo dos de estas presentan buenos resultados. Es decir, las técnicas de mutación por inversión y desplazamiento se destacan sobre los otros dos métodos, mutación por intercambio y revuelto (scrambled). Inclusive, para el caso de 500 ciudades se obtiene un resultado menor al reportado en la literatura; la mejor solución presentada en los benchmark es de 8456 mientras que el obtenido con la implementación de mutación por desplazamiento es de 8448.

La tabla 3 expone los mejores resultados obtenidos con el algoritmo genético secuencial y el paralelo, además de los resultados reportados en la literatura. Se puede observar que con el paralelo se obtienen buenas soluciones en comparación con el secuencial. De los 8 problemas prueba que se tomaron, en cuatro de ellos (10, 20, 100 y 500 ciudades) se alcanzan las soluciones registradas en los benchmark con el paralelo, destacando el resultado obtenido para el problema de 500 ciudades, mientras que con el

secuencial únicamente dos soluciones son idénticas a los reportados en la literatura (10 y 100 ciudades). En el problema de 280 ciudades la diferencia es mínima entre el benchmark y el encontrado con el paralelo, para los problemas restantes, 783, 1002 y 1291 ciudades, la diferencia es considerable entre soluciones halladas con el AG paralelo y los registrados en los benchmark.

De la tabla 4, se observa que para los dos primeros problemas, 10 y 20 ciudades, es suficiente ejecutar el programa con 1 hilo, es decir, basta con ejecutar el programa secuencial debido a que el tiempo de ejecución es menor al programa paralelo ejecutado con distintas cantidades de hilos. Con respecto al problema de 100 ciudades se obtiene un menor tiempo al ejecutarlo con el paralelo y con una cantidad de 10 hilos, para el resto de problemas se logra una mayor eficiencia con relación a la duración de ejecución al correr el programa con 24 hilos. Esto último significa que para los problemas medianos y grandes se logra un mejor rendimiento del hardware al ejecutar el programa con una cantidad de hilos proporcional al número de procesadores, para este caso específico, el nodo de la supercomputadora cuenta con 24 procesadores y es esta misma cantidad de hilos en la cual se alcanzó un mejor rendimiento en relación al tiempo de ejecución.

Hablando en términos globales, se concluye que se encontraron mejoras con el algoritmo genético paralelo esto en relación a los tiempos de ejecución y los resultados que se lograron en comparación con los observados en la literatura. El algoritmo que se tiene codificado hasta el momento funciona perfectamente para los problemas pequeños (10, 20 y 100 ciudades) debido a que se han alcanzado buenas soluciones, sin embargo, para los problemas mayores (783, 1002 y 1291 ciudades) no se han hallado buenos resultados.

Finalmente, como trabajo a futuro se planea implementar un programa híbrido en el cual se pueda trabajar con memoria distribuida y compartida mediante la aplicación de MPI y OpenMP, es decir, trabajar con distintas poblaciones distribuidas en diferentes procesadores y hacer un intercambio de individuos, los mejores preferiblemente, cada cierto número de generaciones.

Referencias

1. Dasgupta S.; Papadimitriou C.; Vazirani U.: *NP-complete problems: Algorithms*. McGraw-Hill Education, pp. 233-237 (2006).

2. OpenMP. http://openmp.org/wp/. Última fecha de acceso 05 de Junio de 2017.

3. Problemas benchmark. https://sites.google.com/site/logisticslaboratory/lecture/tsp-program-benchmark-problems. Última fecha de acceso 27 de Julio de 2016.

4. Andrés López C.; Alberto Mendoza J.; Cuartas E.: Algoritmo para la exploración de todos los valores posibles en el problema del agente viajero (TPS). *Scientia et technica*. Vol. 2, No. 39, pp. 399 (2008).

5. Toro Ocampo E.; Iván Bolaños R.; Granada Echeverri M.: Solución del problema de múltiples agentes viajeros resuelto mediante técnicas heurísticas. *Scientia et technica.* Vol. 19, No. 2, pp. 174 (2014).

6. Guerequeta R.; Vallecillo A.: Ramificación y poda. *Técnicas de diseño de algoritmos.* Servicio de Publicaciones de la Universidad de Málaga, pp. 256. (1998).

7. Guevara Díaz J.: *Mapas Autoorganizativos y el Problema del Agente Viajero, Backpropagation y Reconocimiento de Dígitos.* pp. 15.

8. López E.; Salas O.; Murillo A.: El problema del agente viajero: un algoritmo determinístico usando búsqueda tabú. *Revista de matemática.* Vol 1, pp. 130 (2013).

9. Poveda R.; Gómez J.; León E.: *GRISLAS: 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).

10. Tsai C.; Tseng S.; Chiang M.; Yang C.: *A Fast Parallel Genetic Algorithm for Traveling Salesman Problem*, pp. 241-250 (2010).

11. Miguel Escobar L.; Álvarez Martínez David.; Augusto Romero R.: *Equipo asíncrono de agentes basados en recocido simulado aplicado al problema del agente viajero simétrico*. Scientia et Technica. Año XVI, No 49. (2011).

12. Dorigo M.; Gambardella L.: *Ant colony system: a cooperative learning approach to the traveling salesman problem*, Vol. 1, pp. 53-66 (1997).

13. Haup R. L.; Haup S. E.: *Practical Genetic Algorithms*, John Wiley & Sons, Inc. (1998).

14. Mnemstudio. http://mnemstudio.org/genetic-algorithms-mutation.htm. Última fecha de acceso 09 de Agosto de 2017.

15. Richard K.: *Reducibility Among Combinatorial Problems*. pp. 85-103, (1972).

16. MP-TestData. http://elib.zib.de/pub/mp-testdata/tsp/tsplib/tsp/. Última fecha de acceso 21 de Junio de 2017.

17. TSP. http://www.math.uwaterloo.ca/tsp/concorde.html. /. Última fecha de acceso 21 de Junio de 2017.

18. Universidad de Cádiz. http://knuth.uca.es/moodle/mod/page/view.php?id=3414. Última fecha de acceso 16 de Junio de 2017.

19. TSP. http://www.math.uwaterloo.ca/tsp/problem/index.html. Última fecha de acceso 21 de Junio de 2017.

20. Balderrama Tellez V.H: *Modelo de ruteo para maximizar la cantidad de precios recopilados por el programa "quién es quién en los precios"*. pp. 35. (2013).

21. Aguilar J.; Leiss E.: Elementos Introductorios: *Introducción a la computación paralela*. Graficas Quinteto, pp 10-11. (2004).

22. Hidalgo Pérez J.; Arquero Portero M.; Nogueras Duran J.; Salvador Suz A.: Algoritmos genéticos: *Distribución equilibrada del esfuerzo de cómputo en algoritmos genéticos paralelos.* pp 27. (2005-2006).

23. Arnau Llombart V.: Organización de computadores: *Introducción al paralelismo y rendimiento.* http://www.uv.es/varnau/OC_T4.pdf (2012). Última fecha de acceso 15 de julio de 2017.

24. Díaz G.: Ley de Amdahl-Ley de Moore. *Web del profesor*. http://webdelprofesor.ula.ve/ingenieria/gilberto/paralela/05_LeyDeAmdahlYMoore.pdf. Última fecha de acceso 15 de julio de 2017.

25. Universidad Politécnica de Madrid. https://datsi.fi.upm.es/docencia/Arquitectura_09/Multis/AC_MP_OpenMP.pdf. Última fecha de acceso 09 de Agosto de 2017.

26. LNS, http://www.lns.buap.mx/. Última fecha de acceso 27 de Julio de 2017

27. Lenguaje C. http://www.cprogramming.com. Última fecha de acceso 15 de Julio de 2017.

28. Gauzellino P., Santos J., Zyserman F.: *análisis del comportamiento de algoritmos paralelos de elementos finitos con descomposición de dominio en aplicaciones geofísicas.* Mecánica computacional. Vol. 20, pp. 475. (2001).