UNIVERSIDAD AUTONOMA DE NUEVO LEON
FACULTAD DE INGENIERIA MECANICA Y ELECTRICA
DIVISION DE ESTUDIOS DE POSTGRADO

ALGORITMOS GENETICOS CON EDADES PARA RESOLVER EL PROBLEMA DEL AGENTE VIAJERO

POR

ING. MARTE ALEJANDRO CASTRO FABELA

T E S I S

EN OPCION AL GRADO DE MAESTRO EN
CIENCIAS DE LA ADMINISTRACION
CON ESPECIALIDAD EN SISTEMAS

SAN NICOLAS DE LOS GARZA, N. L. JUNIO DE 1999

ALGORITMOS GENETICOS CON EDADES PARA
RESOLVER EL PROBLEMA DEL AGENTE VIAJERO

M. A. C. F.

UNIVERSIDAD AUTONOMA DE NUEVO LEON
FACULTAD DE INGENIERIA MECANICA Y ELECTRICA
DIVISION DE ESTUDIOS DE POSTGRADO

ALGORITMOS GENETICOS CON EDADES PARA RESOLVER EL PROBLEMA DEL AGENTE VIAJERO

POR

ING. MARTE ALEJANDRO CASTRO FABELA

T E S I S

EN OPCION AL GRADO DE MAESTRO EN
CIENCIAS DE LA ADMINISTRACION
CON ESPECIALIDAD EN SISTEMAS

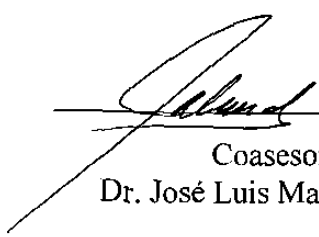
SAN NICOLAS DE LOS GARZA, N. L. JUNIO DE 1999

UNIVERSIDAD AUTÓNOMA DE NUEVO LEÓN
FACULTAD DE INGENIERÍA MECÁNICA Y ELÉCTRICA
DIVISIÓN DE ESTUDIOS DE POSTGRADO

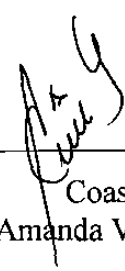
Los miembros del comité de tesis recomendamos que la tesis ALGORITMOS GENÉTICOS CON EDADES PARA RESOLVER EL PROBLEMA DEL AGENTE VIAJERO, realizada por el Ing. Marte Alejandro Castro Fabela, sea aceptada para su defensa como opción al grado de Maestro en Ciencias de la Administración con especialidad en Sistemas.

El Comité de Tesis

Asesor
Dra. Ada Margarita Álvarez Socarrás



Coasesor
Dr. José Luis Martínez Flores



Coasesor
M.C. Amanda Vázquez García

Vo. Bo.
M.C. Roberto Villarreal Garza
División de Estudios de Postgrado

San Nicolás de los Garza, Nuevo León, a 15 de Junio de 1999.

DEDICATORIAS

A Dios:

Por su infinito amor.

Muy Especialmente
a mis Padres:

El señor Felipe de Jesús Castro Sánchez y la señora María Aurelia Fabela de Castro, quienes siempre me han apoyado y me han guiado con sus consejos, ejemplo y amor; los quiero mucho.

A mis Hermanos: *Jesús Osbaldo y Claudia Elizabet por su apoyo y comprensión durante los momentos difíciles. Sigamos siempre unidos.*

A mi Novia:

Nubia Selene García por su cariño, comprensión y apoyo desinteresado, durante el tiempo que hemos estado juntos (TQM).

A mis Amigos:

Por participar conmigo en esta aventura que apenas empieza: la vida.

A todas aquellas personas que de una u otra forma han estado conmigo... en las buenas y en las malas situaciones de la vida.

AGRADECIMIENTOS

Deseo expresar mi más sincero agradecimiento a la Dra. Ada Margarita Álvarez Socarrás, asesora de esta tesis, por su valiosa ayuda y su acertada guía para el desarrollo del presente estudio y por haber contado con su confianza y paciencia para poder concluir esta meta. También quiero agradecer al Dr. José Luis Martínez Flores, coasesor del presente trabajo, por sus consejos y recomendaciones y por brindarme su apoyo para la culminación de esta tesis. De igual forma agradezco a la M.C Amanda Vázquez García por sus revisiones y correcciones necesarias para la realización de este trabajo.

Al Consejo Nacional de Ciencia y Tecnología (CONACYT) por promover y fomentar este tipo de proyectos y por la ayuda económica brindada para la realización de mis estudios. Gracias.

Al Doctorado en Ingeniería de Sistemas (DIS), a la Facultad de Ingeniería Mecánica y Eléctrica (FIME) y a la Universidad Autónoma de Nuevo León (UANL) por permitirme el uso de su equipo e instalaciones.

A todos mis compañeros y amigos por su por su presencia durante estos dos años.

A todas aquellas personas que de una forma u otra contribuyeron en mucho a la realización de este meta.

RESUMEN

Marte Alejandro Castro Fabela **Fecha de Graduación: Junio, 1999**

Universidad Autónoma de Nuevo León

Facultad de Ingeniería Mecánica y Eléctrica

Título del Estudio: **ALGORITMOS GENÉTICOS CON EDADES PARA RESOLVER EL PROBLEMA DEL AGENTE VIAJERO**

Número de Páginas: 83

Candidato para el grado de Maestría en Ciencias de la Administración con especialidad en Sistemas

Área de Estudio: Sistemas

Propósito y Método del Estudio: El propósito principal de esta tesis es abordar el problema del agente viajero (PAV), utilizando para ello, un algoritmo genético con edades (AGE). El PAV consiste en encontrar la longitud mínima de un recorrido, dado un conjunto de n ciudades, de tal manera que partiendo de una cualquiera, se tenga que visitar cada una de ellas exactamente una sola vez y regresar a la ciudad que fue tomada como punto de partida. En 1994 nace un nuevo paradigma dentro de los algoritmos genéticos (AG), el cual tiene como principal característica la introducción del concepto de edades (AGE). Este nuevo enfoque fue probado con algunos problemas de optimización parámetrica y los resultados obtenidos fueron de más alta calidad, en comparación con aquellos que utilizaron un AG tradicional. Al abordar el PAV mediante los AGE se eliminó el proceso de selección, utilizándose en su lugar la estrategia de asignación de edades lineal. Fue necesario introducir una forma de regular el crecimiento excesivo que se produce cuando se utiliza el mecanismo de asignación de edades antes mencionado.

Contribuciones y Conclusiones: Se implementó, por primera vez, un AGE para abordar el PAV, se introdujo una nueva idea que controla el problema del crecimiento desmedido de la población, cuando se utiliza la estrategia lineal de asignación de edades. Se llevaron a cabo pruebas con una instancia en particular del PAV y se compararon con aquellas producidas por un AG estándar. En base a los resultados obtenidos se concluyó que la utilización del AGE para el PAV genera buenos resultados, por lo cual, es conveniente realizar nuevas investigaciones utilizando este enfoque.

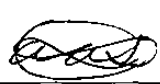
FIRMA DEL ASESOR: 

TABLA DE CONTENIDO

Capítulo **Página**

LISTA DE FIGURAS

Figura **Página**

LISTA DE TABLAS

Tabla **Página**

CAPÍTULO 1

INTRODUCCIÓN

1.1 La evolución y los algoritmos genéticos

En 1858 Charles Darwin y Alfred Wallace presentaron de manera independiente las ideas de la selección natural mediante una explicación científica simple y elegante de la complejidad y la variedad de la naturaleza .

Darwin observó que los organismos (individuos de una especie) generalmente producen mucha descendencia, sin embargo ésta no tiene un crecimiento exponencial, tal como podría esperarse partiendo de esta observación, en vez de ello la población tiende a permanecer en un tamaño constante. Observando que los individuos dentro de una población muestran características propias, concluyó que las fuerzas que actúan sobre la población, incluyendo la competencia entre organismos por alimento o por encontrar pareja, las enfermedades y los depredadores naturales, dan como resultado la desaparición (muerte) de los individuos más débiles y la supervivencia de los individuos más fuertes o aquellos que mejor se adaptaron a su medio ambiente. Así que, los individuos que lograron sobrevivir son los que pueden aparearse y generar una descendencia, la cual hereda las características que ayudaron a sobrevivir a sus

progenitores. La evolución es la acumulación de pequeños cambios positivos en la población a través de la selección natural.

Por otra parte, los algoritmos genéticos, en lo sucesivo AG, son una metodología de resolución de problemas de optimización, que utilizan programas de computadora que emulan algunos de los procesos de la evolución [1,2]. Estos algoritmos, a diferencia de los algoritmos clásicos de optimización, no generan una solución simple en cada iteración, sino que trabajan con un conjunto o población de soluciones que se van mejorando a través de cada iteración (generación). Los AG por lo general son neutrales al contexto, lo cual significa que no consideran ni explotan la información del problema para encontrar la solución al mismo [1]. Al igual que en el proceso de la evolución natural, los AG tratan de producir una descendencia de organismos o individuos (soluciones), que hereden las mejores características de sus antecesores. De esta manera, a partir de un conjunto de soluciones, que se van mejorando a través de cada generación, se pretende encontrar una solución al problema que se está tratando. Aunque los AG no garantizan encontrar la mejor solución al problema, los resultados que genera, para algunos casos, son muy cercanos a la solución óptima.

1.2 El problema del agente viajero

El problema del agente viajero (PAV) a pesar de ser muy sencillo de plantear, ha atraído la atención de numerosos investigadores en los últimos 30 años, dado lo complejo que resulta su resolución. Este problema consiste en determinar, dado un mapa de carreteras, en qué orden deben visitarse n ciudades de tal forma que partiendo de una cualquiera, se recorra el menor número de kilómetros y se vuelva al punto de partida tras visitar una sola vez cada una de ellas [1,3].

Hoy en día se puede obtener una gran cantidad de información acerca de las diferentes metodologías que existen para abordar el problema del agente viajero: búsqueda tabú, recocido simulado, redes neuronales. Un resumen general de ellas se puede encontrar en un documento publicado por Mark H. Noschang en la red internet [3].

1.3 Planteamiento del problema

Los AG han mostrado que pueden encontrar soluciones para el PAV cercanas al óptimo con relativa facilidad. Algunas investigaciones que han utilizado esta metodología proponen diferentes alternativas de llevar acabo su implementación [1,2]. Sin embargo, la mayoría de los estudios realizados se han enfocado en buscar formas de codificar las soluciones del problema (cromosomas) y desarrollar operadores genéticos especiales para estas codificaciones. También se han realizado algunas investigaciones dirigidas hacia el desarrollo de algoritmos híbridos, los cuales son una mezcla de AG y otras metodologías [1,3].

Ahora bien, durante la implementación de un AG estándar en general, existen algunos problemas inherentes al mismo[2]:

- La diversidad de la población. Que consiste en mantener controlada en todo momento la variedad de individuos (soluciones) en la misma. Cuando se presenta este problema (poca diversidad de individuos), se produce como consecuencia una convergencia prematura del AG

- La presión selectiva. La cual se define como la mayor o menor tendencia de asignación de probabilidades de supervivencia para favorecer a los mejores

individuos (aquellos que representan las mejores soluciones del problema). Cuanto más se favorezca a los mejores individuos, se obtendrá menor variedad en la población, por el contrario, si no se favorece a los mejores individuos, aumentará la diversidad en la misma, lo cual producirá que disminuya la eficiencia global del AG.

Ambos temas están relacionados con el tamaño de la población y los métodos de selección utilizados en la implementación de un AG. Las investigaciones realizadas hasta ahora para contrarrestar los problemas antes mencionados, en el caso del PAV, se han dirigido a utilizar algunos métodos de selección que disminuyen en cierta medida este problema [2].

En 1994, Arabas, Michalewicz y Mulawka [2] propusieron una nueva forma de atacar los dos problemas que se mencionaron anteriormente. Dando como resultado el nacimiento del paradigma de los AG con edades. La idea fue tomada de la siguiente analogía de la evolución natural: todos los organismos naturales nacen, se reproducen con mayor o menor éxito, y más tarde o más temprano desaparecen. Este nuevo enfoque fue probado con algunos problemas de optimización parámetrica. Los resultados obtenidos fueron de más alta calidad, en comparación con aquellos que utilizaron un AG tradicional.

Hasta ahora, no se ha realizado un estudio que investigue la eficiencia de los AG con edades en la resolución del PAV. Por consiguiente, resulta interesante realizar una investigación que presente algunos resultados del desempeño del mismo y compararlos contra aquellos que se producen cuando se utiliza un AG estándar.

1.4 Objetivo de la investigación

La presente investigación tiene como principales objetivos:

- Utilizar el enfoque de AG con edades para el PAV. Para ello, se introduce el concepto de edades y la variación del tamaño de la población de individuos.
- Regular el crecimiento de la población, mediante la introducción de una nueva idea que se propone en la presente tesis, la cual fue tomada de una simple observación de la naturaleza.
- Comparar los resultados obtenidos con aquellos que se generan utilizando el enfoque tradicional de los AG (aquel que utiliza algún método de selección).

1.5 Descripción de los capítulos

El capítulo 2 presenta una introducción general a los AG. Se muestra cómo estos simulan algunos procesos de la naturaleza y los utilizan en la resolución de problemas. Se describen las partes que componen un AG y se explica el funcionamiento del mismo. Como parte final de este capítulo de introducción a los AG se presentan algunas de las características distintivas de los mismos.

En el capítulo 3 se exponen algunas de las dificultades de orden teórico y práctico a la hora de implementar un AG. Es importante conocer estas limitaciones e inconvenientes que presentan los AG para poder enfrentarlos; en este apartado se explican algunas de las estrategias que ayudan a contrarrestar dicha problemática.

En el capítulo 4 se presenta el marco teórico para el PAV, se da una explicación general del problema y la formulación matemática del mismo, se presentan algunos conceptos de la Teoría de las Gráficas y cómo se puede formular el PAV a partir de dichos conceptos. Se presenta una breve descripción histórica del problema y se aborda

el tema de la complejidad de los problemas de optimización y a qué clase pertenece el PAV. Se muestran algunas de las formulaciones y aplicaciones del PAV y, para terminar, se presentan algunas de las heurísticas que se pueden utilizar para resolver el PAV.

En el capítulo 5 se exponen las diferentes técnicas que pueden utilizarse para cada uno de los procesos que conforman el AG para, así, resolver el problema del PAV. Este apartado tiene la finalidad de relacionar los dos grandes temas que se presentan en este trabajo de tesis.

En el capítulo 6 se presentan las diferentes pruebas que se llevaron a cabo y los resultados obtenidos que servirán de apoyo a las conclusiones de este estudio.

En el capítulo 7, como punto final de esta tesis, se muestran las conclusiones finales de la misma y se proponen algunas investigaciones futuras que podrían llevarse a cabo.

1.6 Resumen

El objetivo de este capítulo fue el de exponer un visión general del presente estudio. Se presentó una breve introducción de los dos temas principales de esta tesis. Se describió la problemática a tratar y el objetivo de la investigación, además de una descripción específica de cada capítulo.

CAPÍTULO 2

ALGORITMOS GENÉTICOS

2.1 Introducción.

La resolución de problemas basada en la emulación de procesos naturales se hace cada vez más popular. Quizás las metodologías más ampliamente conocidas dentro de este ámbito sean los AG y las Redes Neuronales Artificiales. Los AG fueron desarrollados por John Holland a principios de los años 70s en la Universidad de Michigan. Holland quiso abstraer el proceso de adaptación natural y usarlo en programas de computadoras [2,3,4]. Durante los últimos años ha habido un creciente interés por los AG en una gran variedad de áreas: optimización, teoría de juegos, procesamiento de imágenes, máquinas de aprendizaje, etc. [4,5,6].

2.2 Qué son los Algoritmos Genéticos

2.2.1 La versión biológica

En la naturaleza, los individuos que pertenecen a una población compiten entre

sí para obtener recursos naturales tales como alimento, agua y territorio. También, miembros de la misma especie a diario compiten para atraer a su pareja. Los individuos que tengan más posibilidades de sobrevivir y de reproducirse, tendrán un número relativamente mayor de descendencia, mientras que los individuos que tengan un desempeño pobre tendrán poca o ninguna descendencia. Esto significa que los miembros de una especie que tienden a sobrevivir son aquellos que tienen una mayor capacidad para adaptarse a su medio ambiente [6].

2.2.2 La versión artificial

Los AG hacen uso de analogías directas del comportamiento natural. Ellos trabajan con una población de "individuos", donde cada uno representa una posible solución para un problema específico. A cada individuo se le asigna una puntuación de aptitud de acuerdo a qué tan buena solución es con respecto al problema. Los individuos de más alta aptitud tendrán la oportunidad de "reproducirse", estos serán llamados criadores, y serán sometidos a ciertas transformaciones de alteración y recombinación. De esta manera, se generan nuevos individuos que constituyen la descendencia, la cual comparte ciertas características tomadas de cada progenitor. Los individuos menos aptos de la población tendrán menos probabilidad de reproducirse y en consecuencia irán extinguiéndose [2,6,7].

De acuerdo a lo anterior, una nueva población de posibles soluciones es producida seleccionando los mejores individuos de la "generación" actual, los cuales se aparean para producir el nuevo conjunto de individuos. Esta nueva generación contiene una alta proporción de algunas características obtenidas de los mejores miembros de la generación anterior. De esta forma, después de muchas generaciones, las mejores características son distribuidas a través de toda la población, favoreciendo en cada una de ellas el apareamiento de los individuos más aptos. Las áreas más prometedoras del

espacio de búsqueda son exploradas y, si el AG ha sido diseñado adecuadamente, la población convergerá a una solución satisfactoria para el problema.

El poder de los AG radica en el hecho de que la técnica es robusta [7], y puede ser tratada con éxito en una variedad muy amplia de campos. Aunque este tipo de algoritmos no garantiza encontrar la solución optima global para el problema, generalmente es adecuado para encontrar soluciones aceptables de una manera relativamente rápida [1,6].

2.3 Terminología

Los AG utilizan un vocabulario que es adoptado de la genética. A los individuos que conforman la población se les conoce como *cromosomas* o *genotipos*. Los cromosomas están formados por unidades ordenadas en forma lineal llamadas genes (Figura 1). Cada *gene* tiene una posición dentro de un cromosoma conocida como *loci* y toma algún elemento de un conjunto de posibles valores llamados *alelos* [5,8].

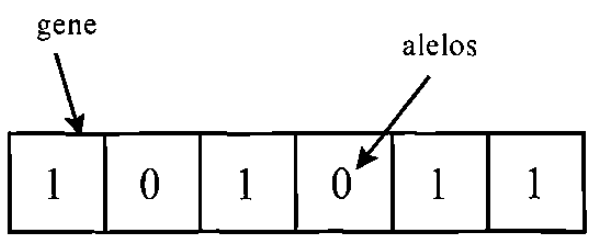


Figura 1. Un cromosoma

2.4 Componentes de un AG clásico

Un AG, independientemente de lo sofisticado que pueda ser su diseño, deberá contener los siguientes 5 componentes [2]:

- Una representación, en términos de "cromosomas", de las soluciones factibles de nuestro problema.

- Una manera de crear la población inicial de las soluciones factibles.

- Una función de evaluación que estime el valor de los individuos en términos de su aptitud.

- Operadores genéticos que permitan alterar la composición de los cromosomas para obtener mejores soluciones.

- Valores de los diferentes parámetros que emplea el AG (tamaño de la población, probabilidades asociadas con la aplicación de los operadores genéticos).

2.4.1 La representación en un AG

En un algoritmo genético típico, la solución de un problema dado se debe representar como un *cromosoma*. Generalmente, éste puede ser una cadena binaria, es decir, cada *gene* puede tomar el valor de 0 o 1(codificación binaria). También, es igualmente factible tener en cada *gene* del cromosoma una representación de un caracter o algún otro número. La longitud del cromosoma representa la longitud de la solución del problema [9,10].

2.4.2 La Población inicial de un AG

La población inicial de un AG se puede producir de diferentes maneras. Por una parte, hacer evolucionar una población de individuos que ha sido generada aleatoriamente, es una forma adecuada de probar el desempeño del AG. Esto es debido a que la solución final debe ser consecuencia del proceso evolutivo del AG y no de las características de los métodos utilizados para generar la población inicial [3]. Por otro lado, para algunas aplicaciones, especialmente las de la industria, puede ser más conveniente inicializar la población del AG utilizando métodos más directos. Una técnica que generalmente se adopta para ello es utilizar un algoritmo goloso. Este es un método heurístico constructivo [3], el cual construye paso a paso la solución buscando el máximo beneficio en cada etapa.

2.4.3 La aptitud en un AG

La calidad de un cromosoma dado está determinada por su aptitud. Ésta es simplemente un valor numérico que se calcula por una función de evaluación específica del problema, la cual es diseñada por el programador. Por ejemplo (figura 2 y 3), la aptitud de un individuo en un problema donde la meta sea maximizar el número de 1s que contiene en su estructura, podría ser simplemente la cantidad de 1s que están presentes en ese cromosoma [6,9].

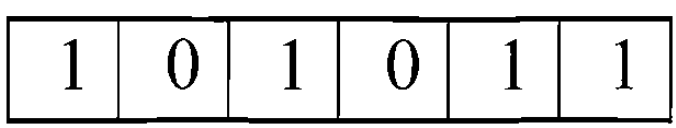


Figura 2. Aptitud = 4

1	0	0	0	1	0

Figura 3. Aptitud = 2

2.4.4 La selección en un AG

Después de haber decidido qué tipo de codificación usaremos, el paso siguiente es elegir cómo llevar a cabo la selección. El proceso de selección consiste en muestrear, a partir de la población actual los elementos de la población de progenitores. Como los individuos de la población de progenitores heredan sus genes a la siguiente generación, es deseable que ésta esté conformada por elementos de alta aptitud [11]. Los mecanismos de muestreo son muy variados, distinguiéndose tres grupos fundamentales según el grado de aleatoriedad del proceso [7]:

1. Muestreo directo: se toma un subconjunto de individuos de la población siguiendo un criterio fijo, del estilo de "los k mejores", "los k peores", "por reglas de dedo", etc.

2. Muestreo aleatorio simple o equiprobable: Se asignan a todos los elementos de la población base las mismas probabilidades de formar parte de la muestra y se constituye ésta mediante ensayos de Bernoulli simples.

3. Muestreos estocásticos: Se asignan probabilidades de selección o puntuaciones a los elementos de la población base en función (directa o indirecta) de su aptitud. La puntuación p_i asociada al individuo x_i de la población $P = \{x_1,...,x_n\}$ se calcula como la aptitud relativa de dicho individuo; esto es, siendo $u_1,...,u_n$, las respectivas aptitudes se tiene que

$$p_i = \frac{u_i}{u_1 + \ldots + u_n} \qquad (\forall i = 1,\ldots,n)$$

Así se garantiza que $p_1 + \ldots + p_n = 1$, tal como corresponde a una distribución de probabilidades. La muestra se constituye de modo estocástico haciendo uso de dichas puntuaciones.

2.4.5 Mecanismos de muestro estocástico

Existen muchos mecanismos de muestreo estocástico según sea la aplicación. En concreto, al implementar un algoritmo genético se usan fundamentalmente cuatro tipos: por sorteo, por restos, universal o por ruleta, y por torneos [7]. A continuación se describe el muestreo por sorteo, que debido a sus buenas propiedades teóricas, es el que se utiliza por lo general con los AG.

Para ilustrarlo, se considerará el problema de muestrear estocásticamente dos elementos de la población $P = \{x_1, x_2, x_3, x_4\}$, siendo las puntuaciones asociadas $p_1 = 0.5$, $p_2 = 0.3$, $p_3 = 0.1$, $p_4 = 0.5$ respectivamente.

1. Calcular las puntuaciones acumuladas

$$q_0 := 0$$
$$q_i := p_1 + \ldots + p_i \qquad (\forall i = 1,\ldots,n)$$

2. Generar k números aleatorios simples $r_j \leftarrow URand[0,1)$ $(\forall j = 1,\ldots,k)$ donde k representa el tamaño de la muestra.

3. Para cada $j = 1,...,k$ se elige el individuo x_i que verifique

$$q_{i-1} < r_j < q_i .$$

Para el ejemplo propuesto, las puntuaciones acumuladas son:

$$q_1 = 0.1 \quad q_2 = 0.4 \quad q_3 = 0.5 \quad q_4 = 1.0$$

respectivamente; y los $k = 2$ números aleatorios simples son los siguientes:

$$r_1 = 0.02433 \text{ y } r_2 = 0.51772 ,$$

de esta manera, el primer elemento de la muestra resulta ser x_1, dado que $q_0 < r_1 < q_1$ y el segundo será x_4, puesto que $q_3 < r_2 < q_4$.

2.4.6 Los Operadores de Cruzamiento

Los operadores de cruzamiento actúan sobre parejas de individuos (operadores binarios) y normalmente originan otro par de individuos que heredan la combinación de algunas características de los progenitores [8,12]. Existen muchos tipos de cruzamiento, en el presente trabajo solo ilustraremos tres formas alternativas.

1. Cruzamiento de un Punto. El procedimiento del cruzamiento de un punto es el siguiente:

1. Generar aleatoriamente un número (menor o igual que la longitud del cromosoma) como la posición de cruce.

2. Conservamos sin cambiar los bits antes del número e intercambiamos los bits después del punto de cruce entre los dos progenitores (figura 4).

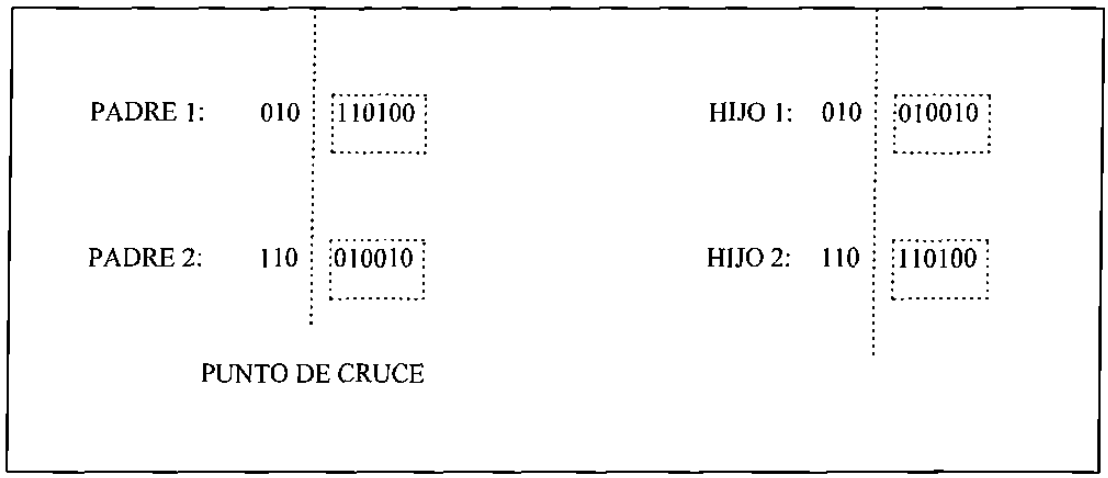


Figura 4. Cruzamiento de un punto

2. Cruzamiento de dos Puntos. Este tipo de cruzamiento es similar al de un punto, excepto que debemos seleccionar dos posiciones y sólo los bits intermedios serán intercambiados. Este tipo de cruce preserva la primera y la ultima parte de un *cromosoma* y sólo intercambia el segmento intermedio (figura 5).

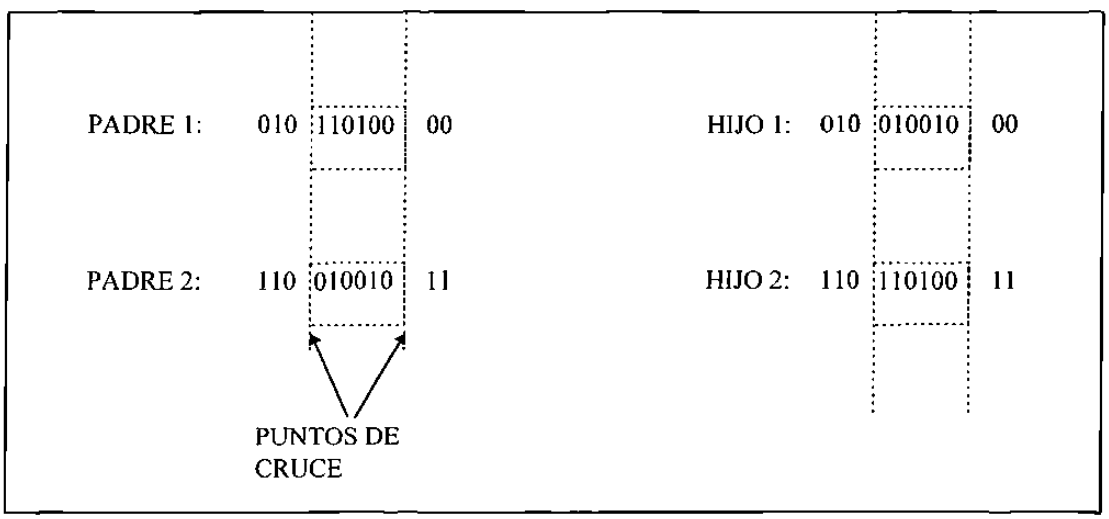


Figura 5. Cruzamiento de dos puntos

3. Cruzamiento Uniforme. El procedimiento del cruzamiento uniforme (figura 6) es el siguiente:

1. Generar un número aleatorio (n), el cual indica cuántas posiciones intercambiar.

2. Generar aleatoriamente n números $\{P_1, P_2, ..., P_n\}$, los cuáles indicarán qué posiciones intercambiar. Para cada P_i intercambiar el $P_{i-esimo}$ bit de un progenitor con el $P_{i-esimo}$ bit del otro, de esta manera se generan dos nuevos hijos.

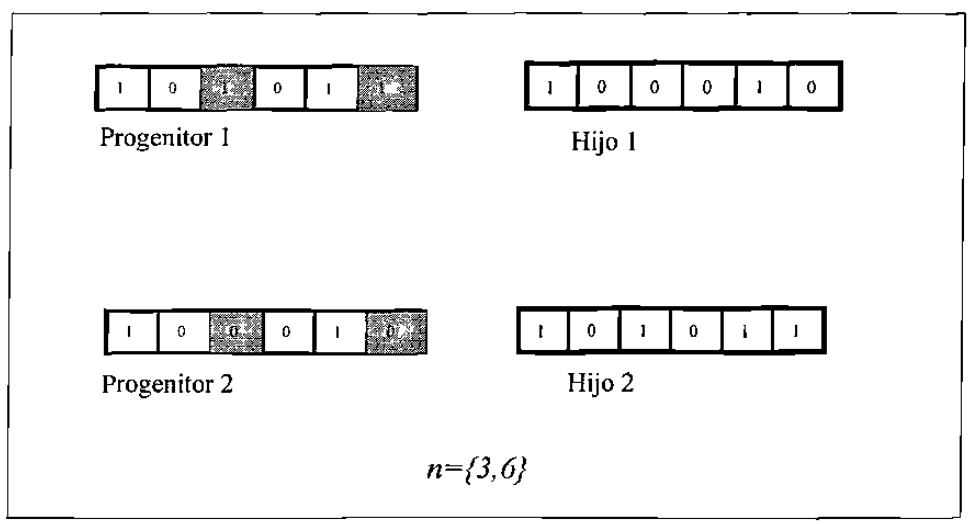


Figura 6. Cruzamiento uniforme

2.4.7 El Operador de Inversión de un AG

La inversión es un operador unitario (sólo trabaja con un individuo) que selecciona dos puntos de cruce en una cadena (*cromosoma*) e invierte el orden de las

posiciones intermedias recordando el contenido de cada una. Esto último obliga a identificar los bits en dichas cadenas, lo cual se lleva a cabo colocando a cada alelo una etiqueta con su posición [7]. Por ejemplo,

$$\langle 0_1 0_2 0_3 | 1_4 1_5 0_6 1_7 | 0_8 0_9 0_{10} 1_{11} \rangle,$$

los puntos de cruce se representan mediante el símbolo "| ", entre los puntos señalados se obtiene

$$\langle 0_1 0_2 0_3 | 1_7 0_6 1_5 1_4 | 0_8 0_9 0_{10} 1_{11} \rangle.$$

Muchos investigadores han utilizado el operador de inversión en su trabajo, aunque parece ser que pocos han intentado justificarlo, o cuantificar su contribución.

2.4.8 El Operador de mutación

La mutación es un operador unitario que provee una forma útil de introducir información en un cromosoma que ha estado perdiendo información durante el cruzamiento. La mutación es simplemente una alteración aleatoria ocasional de un gene dentro del cromosoma. La mutación puede ser valiosa, pero es generalmente considerada como de segunda importancia en el proceso general del algoritmo genético. El porcentaje de mutación es típicamente del orden de 1 gene en 100, esto se debe a que un porcentaje de mutación alta podría simplemente reducir el problema a una búsqueda al azar.

2.4.9 La convergencia de un AG

Cuando el AG se ha implementado correctamente, la población evolucionará a través de generaciones sucesivas con el fin de que la aptitud del mejor individuo y el promedio de aptitudes general dentro de cada generación se dirija hacia el óptimo global. La convergencia es el avance hacia una uniformidad creciente. Se dice que la población ha convergido cuando todos los individuos comparten el mismo valor [6]. La Figura 7 muestra cómo varía la aptitud en un AG. Se puede observar que mientras la población converge, la aptitud promedio se aproximará al mejor individuo.

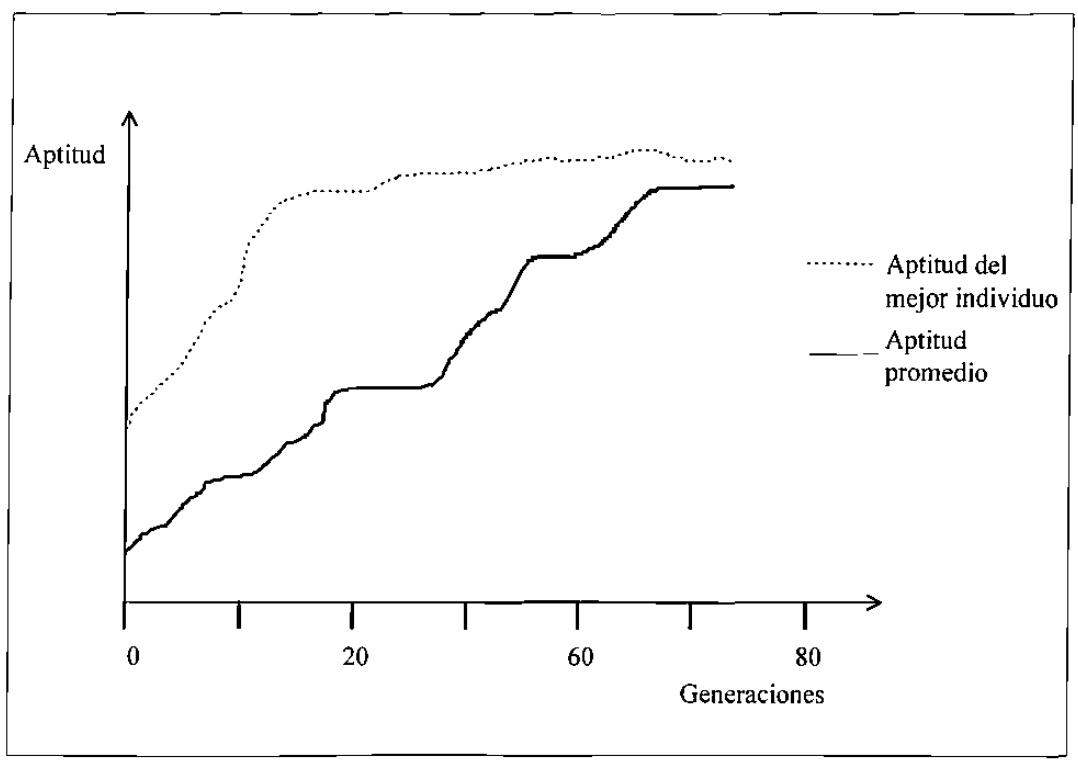


Figura 7. La convergencia de un AG

2.5 Funcionamiento de un AG clásico

Un AG se puede implementar en formas ligeramente diferentes, pero el bosquejo es casi siempre el mismo [8].

1. Inicializa y codifica una población de cromosomas al azar.

2. Decodifica y evalúa la aptitud de cada cromosoma en la población.

3. Reproduce una nueva generación, para ello selecciona los cromosomas que actuarán como progenitores, de acuerdo a su aptitud, para procrear una descendencia nueva.

4. Aplica los operadores de cruzamiento y mutación a los nuevos cromosomas.

5. Los puntos 2 al 4 se repiten hasta completar un número determinado de iteraciones (generaciones).

2.6 Características distintivas de los AG

Los AG son métodos de búsqueda:

- **ciega**, es decir, no disponen de ningún conocimiento específico del problema, de manera que la búsqueda se basa exclusivamente en los valores de la función objetivo.

- **codificada**, puesto que no trabajan directamente sobre el dominio del problema, sino sobre representaciones de sus elementos.

- **múltiple**, esto es, buscan simultáneamente entre un conjunto de candidatos.

- **estocástica**, referida tanto a las fases de selección como a las de transformación.

2.7 Resumen

Los AG son "algoritmos de búsqueda que están basados en mecanismos de la genética y selección natural". Durante el desarrollo de este estudio se ha proporcionado una visión general de los AG, se presentaron los conceptos básicos y se describió el funcionamiento de los mismos con el propósito de establecer una marco teórico para los siguientes capítulos.

CAPÍTULO 3

TÓPICOS SELECTOS DE AG

3.1 Introducción

En el presente capítulo se plantean varias dificultades de orden teórico y práctico que se presentan a la hora de llevar a cabo una implantación del modelo de AG para resolver problemas de optimización. Todas ellas surgen como consecuencia de cuatro limitaciones intrínsecas de dicho modelo; es por ello que para corregir o mitigar sus efectos conviene estudiar antes con cierto detenimiento las implicaciones generales de tales limitaciones. De esta manera, se podrá emprender el diseño de procedimientos eficaces de resolución.

3.2 Limitaciones de los AG e inconvenientes asociados

De modo general, las cuatro limitaciones del AG básico (AGB) son [7]:

1. La representación: El espacio de búsqueda sólo puede ser representado mediante cadenas binarias.

2. La irrestricción: El mecanismo del AGB no considera las posibles restricciones o ligaduras que pudieran imponerse a la búsqueda.

3. La opacidad: El AGB es un algoritmo de búsqueda ciega, esto es, sólo está guiado por la aptitud de los individuos , y no incorpora ningún otro conocimiento específico del problema en cuestión.

4. La finitud: A efectos prácticos, el AGB sólo puede trabajar con poblaciones finitas no muy grandes.

Los inconvenientes que ocasionan las dos primeras limitaciones no están fuera de lo previsible, sin embargo, la opacidad y la finitud son causa original de otros muchos inconvenientes que se tendrán que atacar por separado. De manera esquemática, puede decirse que todos estos inconvenientes tienen su base en dos hechos [7]:

1. Fundamentalmente, la opacidad es causa inevitable de debilidad.

2. Por otra parte, la opacidad acentúa fuertemente los problemas de diversidad derivados de la finitud de la población.

A continuación se estudiarán más detenidamente sus consecuencias.

3.2.1 El problema de la debilidad de los AG

El inconveniente más visible de la debilidad es la ineficiencia: los métodos débiles de resolución de problemas presentan las ventajas de ser muy poco específicos, poco exigentes y notablemente eficaces, sin embargo, todos ellos son ineficientes en mayor o menor grado.

Otro de los inconvenientes prácticos de la debilidad es la tendencia al extravío de la búsqueda [7]. Este fenómeno se puede explicar de la siguiente manera: el AGB realiza la búsqueda de los mejores puntos utilizando únicamente la aptitud de los individuos; en ocasiones ocurre que la información proporcionada (ie., la aptitud) resulta insuficiente para orientar correctamente al algoritmo en su búsqueda del óptimo. A esto se le llama desorientación. En el peor de los casos puede ocurrir que este fenómeno tenga tanta fuerza como para impedir que el AG converja al óptimo global, desviándolo finalmente hacia un óptimo local. Esto es el extravío propiamente dicho. Afortunadamente, aunque un AG se desoriente no necesariamente se va a extraviar. Para ello es necesario partir de una mala población inicial o plantear un problema especialmente diseñado para que se extravíe.

Conviene señalar que la presencia de desorientación suele estar fuertemente estimulada por una mala codificación que "oculte" la ya de por sí escasa información disponible [7]. De modo recíproco, una buena codificación reduce la probabilidad de extravío. Una explicación concisa de la codificación que se utiliza para acometer el problema que se trata en este estudio, se presenta en el capítulo 5.

3.2.2 El problema de la diversidad de los AG

Para efectos prácticos, es fundamental que todo AG verifique lo siguiente:

Requisito de variedad en los AG [7]: Es esencial, para el buen funcionamiento de un AG, controlar en todo momento la diversidad de la población. Se entiende la diversidad en sentido general como "diversidad de individuos" y en particular como "diversidad de aptitudes".

3.3 Convergencia prematura por problemas de diversidad

Durante la ejecución de un AG puede ocurrir que un individuo, o un grupo de ellos, obtenga una aptitud notablemente superior a los demás. Esto es especialmente probable en las fases tempranas de la evolución, en las cuales, por la aplicación de los operadores genéticos, suele surgir un buen candidato de entre una población de individuos mediocres [6,7,9]. En tal circunstancia, existe el riesgo de que se produzca una *evolución en avalancha*: al incrementar los individuos más aptos su presencia en la población, por ser ésta finita la diversidad disminuye, ello hace que en la siguiente generación se favorezca aún más a los individuos más aptos hasta que estos acaban dominando por completo la población. A esto se le conoce como el fenómeno de los *superindividuos*. Habitualmente ocurre que tales *superindividuos* sólo son los más aptos en cierto momento, pero no los más aptos absolutos, téngase en cuenta que la falta de diversidad estanca la búsqueda, lo que en último término provoca una *convergencia prematura* del AG, habitualmente hacia un subóptimo.

La única circunstancia en que este fenómeno es tolerable e incluso deseable es en las fases tardías de la evolución, cuando el AG ha "localizado" correctamente la solución óptima, pero nunca antes.

3.4 Mecanismos de control de diversidad

3.4.1 Presión selectiva

Para controlar la diversidad de las aptitudes se debe actuar sobre el mecanismo de asignación de probabilidades de supervivencia (o de descendientes, en su caso): cuanto más se favorezca a los más aptos menor variedad se obtendrá y, como consecuencia, aumentará el riesgo de perder la información de valor desarrollada por los individuos moderadamente aptos, entendiendo por "información de valor" toda la que podría servir

posteriormente para hallar el óptimo global. Por el contrario, si no se favorece especialmente a los individuos más aptos, se obtendrá más diversidad en la población, pero a costa de hacer las etapas de selección mucho menos eficaces, lo que empeora la eficacia global del AG [2,7].

A la mayor o menor tendencia del mecanismo de asignación de probabilidades de supervivencia para favorecer a los individuos más aptos, se le llama *presión selectiva* [6,7]. Lo ideal sería que en las primeras fases de la evolución de un AG hubiera poca *presión selectiva* con el fin de que la búsqueda fuera lo más global posible y en las últimas fases de la evolución, una vez que ya ha sido "localizada" la mejor solución, se incrementara fuertemente la *presión selectiva* para encontrarla cuanto antes.

A efectos prácticos, se puede controlar la diversidad de aptitudes de cuatro maneras [7]:

1. Actuando directamente sobre el mecanismo de selección: Se trata de hacerle lo más insensible posible a la distribución de aptitudes y a los errores de muestreo, con la idea de evitar la convergencia prematura. Esto se consigue usando mecanismos de muestreo distintos al de por sorteo.

2. Actuando sobre el mecanismo de asignación de aptitudes: Se trata de conseguir una distribución de aptitudes lo más uniforme posible, modificando apropiadamente el mecanismo de asignación de aptitudes. Así, la probabilidad de supervivencia deja de ser necesariamente proporcional a la aptitud bruta.

3. Controlando las "edades" de los individuos.

4. Introduciendo mecanismos de escisión.

Durante el desarrollo de este capítulo se explicarán los puntos 1 y 3. Para un estudio dctallado de los puntos 2 y 4 se pueden consultar los trabajos independientes de Zbigniew Michalewicz, David E. Goldberg, y A. Pérez Serrada [2,4,7].

3.5 Mecanismos de muestreo

Como se mencionó anteriormente existen dos importantes temas en el proceso de evolución de la búsqueda genética: la diversidad en la población y la presión selectiva. Estos dos factores están íntimamente relacionados: un incremento en la presión selectiva disminuye la diversidad en la población y viceversa. En otras palabras, una presión selectiva fuerte conlleva hacia una convergencia prematura del AG; una presión selectiva débil produce una búsqueda inefectiva [2]. Es por ello, que se necesita encontrar un punto de equilibrio entre estos dos factores. Mediante los mecanismos de muestreo se intenta alcanzar este objetivo. A continuación se describen algunos de ellos.

3.5.1 Muestreo estocástico universal o de ruleta

Es análogo al muestreo por sorteo, sólo que ahora se genera un único número aleatorio simple r y con él se asignan todas las muestras de modo parecido a como se haría al girar una ruleta.

Para simular una tirada de ruleta se definen, a partir de $r \leftarrow URand[0,1)$, los siguientes números:

$$r_j := \frac{r + j - 1}{k} \qquad \forall j = 1,\ldots,k$$

Con esto ya se puede determinar la muestra, comparando los r_j con las puntuaciones acumuladas tal como se mostró en el capítulo 2.

3.5.2 Muestreo por valor esperado

El muestreo por valor esperado es una variante del muestreo por sorteo [2]. Éste consiste en asignar a cada uno de los individuos unos contadores que se inicializan en $\lceil 2 \cdot p_i \rceil$, donde p_i, como se mencionó en el capítulo 2, representa la probabilidad de selección o puntuación en cada individuo. Cada vez que un individuo de la población es elegido para reproducirse, se decrementa el contador en 1 ó en 2, según el operador de reproducción empleado (habitualmente 1 para el cruce y 2 para la mutación); cuando el contador de un individuo llegue a ser cero, se le excluye de posteriores muestreos.

3.5.3 Selección por torneos

Cada elemento de la muestra se toma eligiendo el mejor de los individuos de un conjunto de z elementos tomados al azar de la población base, esto se repite k veces hasta completar la muestra. El parámetro z suele ser un entero pequeño comparado con el tamaño de la población base, normalmente 2 ó 3. Nótese que para este caso, no es necesario hacer la asignación de puntuaciones [6,7].

3.6 AG con edades

Todos los organismos naturales nacen, se reproducen con mayor o menor éxito y más tarde o más temprano mueren. A partir de esta perspectiva se introduce un concepto

relativamente nuevo en los AG: los algoritmos genéticos con edades, en lo sucesivo AGE, el cual se puede explicar de la siguiente manera: en el momento de ser creado un individuo se le asigna una duración dependiendo de su aptitud y se pone a cero un contador de edad; tras cada generación se incrementa la edad de cada individuo y se eliminan aquéllos que hallan rebasado su duración.

Más allá de discusiones teóricas sobre la emulación de la naturaleza, la introducción de la edad en los individuos de los AG está justificada por dos motivos concretos [7]:

1. Ese mecanismo se basta para controlar la presión selectiva, de modo tal que hasta hace superflua la fase de selección de progenitores, lo que evita en principio todos los inconvenientes asociados a ella.

2. La introducción de la edad obliga a hacer variable el tamaño de la población, lo cual resulta ser una ventaja más que un inconveniente, pues proporciona un mecanismo de autorregulación de dicho tamaño.

Ambas cuestiones resultan ser de máximo interés: por un lado, el concepto de edad del cromosoma reemplaza a la selección y con ello a todos los posibles inconvenientes derivados de este tipo de procesos. Por otro lado, el tamaño de la población es una de las elecciones más importantes a la hora de sintonizar (encontrar los parámetros adecuados) un AG, pudiendo ser crítica en muchas aplicaciones: si el tamaño de la población es demasiado pequeño tendrá una marcada tendencia a la convergencia prematura, y si es demasiado grande desperdiciará recursos computacionales sin obtener mejoras apreciables.

El AGE procesa en un tiempo t la población $P(t)$ de cromosomas. Durante el paso de reproducción de $P(t)$, se genera una población auxiliar (esta población contiene la descendencia). El tamaño de la población auxiliar es proporcional al tamaño de la población original: $AuxPopSize(t) = \lfloor PopSize(t) \cdot \rho \rfloor$ cromosomas (el parámetro ρ se

conoce como razón de producción) [2]. Cada miembro de la población puede ser elegido para reproducirse (colocar la descendencia en la población auxiliar) con probabilidad idéntica independientemente de su valor de aptitud. La descendencia se crea aplicando los operadores genéticos (cruzamiento y mutación) a los cromosomas seleccionados. Como puede observarse, la elección de los cromosomas no depende del valor de aptitud, no existe un proceso de selección, se introduce el concepto de *edad* del cromosoma y su parámetro *tiempo de vida* (duración).

A cada individuo se le asigna su duración al ser evaluado por primera vez y dicho parámetro permanece constante a lo largo de todo el proceso de evolución, hasta que el individuo desaparece (muera), lo cual sucede cuando éste ha superado su duración. Es por ello, que el tamaño de la población tras la *t-ésima* generación es

$$PopSize(t+1) = Popsize(t) + AuxPopSize(t) - D(t),$$

donde *D(t)* es el número de individuos que rebasaron su duración en dicha generación.

El punto más delicado del AGE es el criterio de asignación de duraciones a los individuos. Para ejercer un control eficaz sobre la presión selectiva, es necesario que dicho criterio [2,7]:

1) refuerce la presencia de los individuos más aptos y reduzca la presencia de los menos aptos,

2) controle el tamaño de la población de modo que no sea tan pequeño que ocasione una perdida de diversidad, ni tan grande que se desperdicie esfuerzo computacional o, peor aún, conlleve a un crecimiento incontrolado de la población.

La práctica muestra que a la hora de calcular la duración de un individuo conviene considerar, además de su aptitud, el estado presente de la búsqueda. Éste se mide a través de los siguientes parámetros: $AvgFit$, $MaxFit$, $MinFit$, los cuales representan los valores de aptitud media, mínima y máxima en la población actual respectivamente, mientras que $AbsFitMax$ y $AbsFitMin$ representan respectivamente los valores de aptitud máxima y mínima absolutos. Teniendo como base lo anteriormente planteado, a continuación se presentan tres estrategias de carácter general para asignar la duración de un individuo [2,7]:

1. **Asignación proporcional:**

$$LifeTime(v) = min\{MaxLT, MinLT + DifLT\frac{fitness(v)}{AvgFit}\}$$

2. **Asignación lineal:**

$$LifeTime(v) = MinLT + 2DifLT\frac{fitness(v) - AbsFitMin}{AbsFitMax - AbsFitMin}\}$$

3. **Asignación bilineal:**

$$LifeTime(v) = \begin{cases} 1)\, MinLT + DifLT\dfrac{fitness(v) - MinFit}{AvgFit - MinFit}, \\ 2)\, \frac{1}{2}(MinLT + MaxLT) + DifLT\dfrac{fitness(v) - AvgFit}{MaxFit - AvgFit} \end{cases}$$

Se utiliza 1 si $Fitness(v) \leq AvgFit$, y 2 si $Fitness(v) > AvgFit$

Los parámetros *MaxLT* y *MinLT* representan respectivamente el tiempo de vida máximo y mínimo permitido. Estos valores se dan *a priori* siendo por lo general 7 y 1 respectivamente. Por otra parte, se define

$$DifLT := \frac{1}{2}(MaxLT - MinLT)$$

La primera estrategia tiene cierto parecido con la selección por sorteo, en el sentido de que las probabilidades de supervivencia de un individuo son proporcionales a su aptitud relativa (dentro de los límites *MinLT* y *MaxLT*).

La experiencia muestra que conviene considerar también la aptitud referida no a la aptitud media, sino a la mejor aptitud obtenida hasta entonces. Esta observación motiva la estrategia de asignación lineal, con lo cual se mejoran las prestaciones de la asignación lineal.

La estrategia de asignación lineal presenta el inconveniente de que cuando hay muchos individuos con aptitudes similares a la del individuo de más alta aptitud, se asignan largas duraciones a todos ellos, lo que hace crecer fuertemente el tamaño de la población y con esto el costo computacional. Con la estrategia de asignación bilineal se trata de rebajar esta tendencia acentuando las aptitudes próximas a la mejor [7].

3.7 Resumen

Hasta este punto se da por terminado el marco teórico de los AG. Durante el desarrollo de este capítulo se presentaron algunas dificultades o limitaciones a la hora de implementar un AG. Después de analizar dicha problemática, se describieron algunas alternativas de solución. Conviene señalar que algunas de estas alternativas se implementarán y se analizará el desempeño de las mismas durante el capítulo de pruebas.

CAPÍTULO 4

EL PROBLEMA DEL AGENTE VIAJERO

4.1 Introducción

Si un vendedor inicia un viaje (recorrido) en la ciudad donde vive y tiene que visitar cada una de las ciudades, de una lista de su itinerario, exactamente una vez y regresar a casa, él podría elegir una ruta o recorrido al azar. No obstante, si el vendedor quiere ahorrar tiempo y energía, por no mencionar el dinero de la gasolina, él tratará de encontrar la ruta más corta con la cual pueda cumplir el objetivo de su trabajo. Si el agente viajero conoce la distancia que existe entre cada par de ciudades, puede encontrar el camino más corto de su viaje sumando las distancias de cada ruta posible y después compararlas. Sin embargo, cuando el número de ciudades aumenta, el tiempo que se requiere para calcular la ruta más corta también se incrementa. La cuestión de encontrar el recorrido más corto, ha sido bautizada con el nombre de *El Problema del Agente Viajero* (PAV).

4.2 Formulación del PAV

Sean dado un conjunto finito $C = \{c_1, c_2, ...c_m\}$ de ciudades y, para cada par de ciudades c_i, c_j en C, la "distancia" $d(c_i, c_j)$ entre ellas. Una solución es un ordenamiento $< c_{\pi(1)}, c_{\pi(2)}, ..., c_{\pi(m)} >$ de las ciudades dadas que minimice

$$\left[\sum_{i=1}^{m-1} d(c_{\pi(i)}, c_{\pi(i+1)})\right] + d(c_{\pi(m)}, c_{\pi(1)})$$

Esta expresión da la longitud del "recorrido" que empieza en $c_{\pi(1)}$, visita cada ciudad en orden, y después regresa directamente a $c_{\pi(1)}$ desde la última ciudad $c_{\pi(m)}$ [13].

Una instancia del PAV, ilustrada en la figura 8 está dada por $C = \{c_1, c_2, c_3, c_4\}$, $d(c_1, c_2) = 10$, $d(c_1, c_3) = 5$, $d(c_1, c_4) = 9$, $d(c_2, c_3) = 6$, $d(c_2, c_4) = 9$ y $d(c_3, c_4) = 3$. El ordenamiento $< c_1, c_2, c_3, c_4 >$ es una solución para esta instancia, con una longitud de 28, mientras que el recorrido correspondiente a la mínima longitud posible es $< c_1, c_2, c_4, c_3 >$ con una distancia de 27 [13].

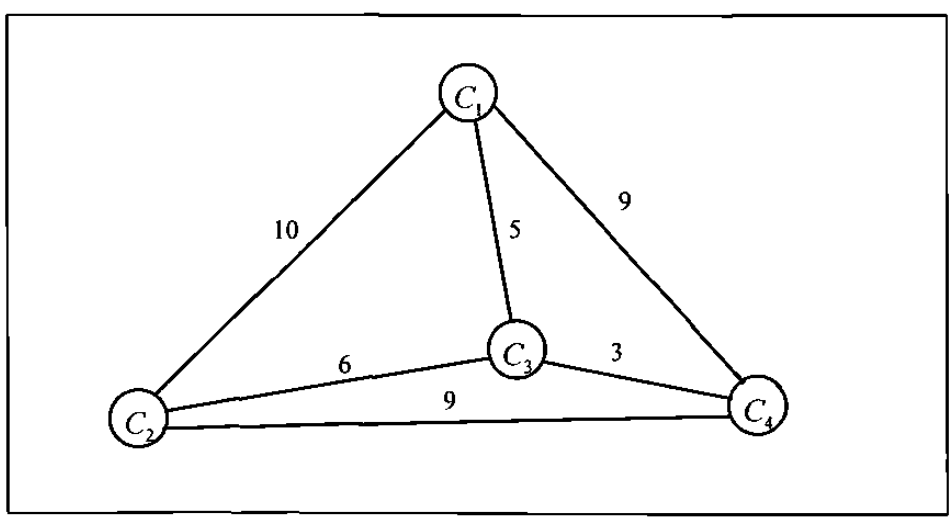


Figura 8. Instancia del PAV

4.3 Definición formal del PAV

Dada un gráfica completa no dirigida $G = (V, E, W)$, en la que V es un conjunto de vértices, E es un conjunto de aristas y W es un conjunto de costos asociados a las aristas (cada arista tiene un costo), el problema consiste en encontrar un ciclo de Hamilton (una ruta cerrada que pase una y sólo una vez por cada ciudad) cuyo costo (la suma de los costos de las aristas por las que pasa el ciclo de Hamilton) sea mínimo [3,14,15].

En ciertas ocasiones se acostumbra definir a la gráfica como un conjunto de ciudades, donde cada una ocupa una posición en un espacio; los costos de las aristas se obtienen de las distancias entre las ciudades en dicho espacio.

4.4 Historia del PAV

El PAV es un problema relativamente antiguo, el cual fue documentado en 1759 por Euler cuyo interés fue resolver un problema llamado el recorrido del caballo. Una solución correcta se obtendría cuando el caballo visitara cada uno de los 64 cuadros de un tablero de ajedrez, exactamente una vez en su recorrido [2,3].

El término agente viajero fue usado por primera vez en el año de 1932 en un libro alemán escrito por un agente viajero veterano: *"El agente viajero, cómo y qué debe hacer para obtener comisiones y tener éxito en su negocio"* [2]. Aunque no fue el tema principal del libro, el tema del PAV y la planificación se discuten en el último capítulo.

El PAV se introdujo por la corporación RAND en 1948 [2]. La reputación que tenía esta asociación ayudó a que el PAV fuera un problema muy conocido y popular. Además el PAV comenzó a ser renombrado en ese tiempo debido al nuevo tema de programación lineal y a los intentos de resolver problemas combinatorios.

4.5 Complejidad del PAV

El PAV pertenece al tipo de problemas *NP completo* [13]. Consecuentemente, cuando el número de ciudades es grande no existen métodos de solución eficientes que garanticen encontrar la ruta óptima. Se puede desarrollar un algoritmo de ejecución rápida, o uno que encuentre la mejor ruta, pero no es posible hacer alguno que realice ambas cosas. A consecuencia de ello, los métodos de solución que se emplean generalmente, solo encuentran una solución casi óptima en un tiempo razonable [2,3,16].

4.6 Formulaciones del PAV

Al principio de este capítulo se describió el PAV en forma general, sin embargo, existe un número de problemas de optimización combinatoria que se relacionan con el mismo, los cuales son modificaciones pequeñas del PAV general. A continuación se presentan algunas formulaciones de estos problemas:

- El PAV asimétrico: Para la formulación de este problema, el costo de viajar de una ciudad i a una ciudad j no es necesariamente igual que hacerlo de manera inversa [3].

- El problema del multiagente viajero: En lugar de considerar solamente un vendedor, se toman m vendedores que están localizados en una ciudad y su meta es visitar las ciudades restantes. La idea es asignar rutas a cada vendedor de tal forma que la combinación de todos sus recorridos cumplan con el requisito de visitar cada ciudad exactamente una vez y se minimice el costo total de los recorridos [3].

- El problema del agente viajero métrico-angular: Dado un conjunto de puntos en el espacio euclideano, encontrar un recorrido que minimice el costo angular total del mismo. El costo angular de la ruta es la suma de los cambios de dirección en los puntos (vértices). Esta formulación surgió debido a las necesidades de las aplicaciones en Robótica [3].

4.7 Aplicaciones practicas del PAV.

El PAV es un modelo computacionalmente importante, pues constituye un componente central de modelos más complicados. Como se mencionó anteriormente, hay un número de problemas del mundo real que se pueden formular como un PAV. Algunas de las aplicaciones que más se conocen son las siguientes:

- Fabricación de circuitos integrados VLSI. El problema es minimizar el tiempo que se requiere al grabar una secuencia de líneas (aristas) que forman las conexiones eléctricas entre los diferentes componentes de un circuito integrado. Para ello, se utiliza una máquina que graba las líneas según el diseño. Ésta se mueve a la posición donde principia la interconexión y traza una línea, terminando en una posición diferente, para después continuar con la siguiente, y así sucesivamente. De esta manera, los puntos de inicio y final del grabado corresponden a las ciudades en el PAV [2,3].

- Tableros de circuitos perforados. Para permitir el acoplamiento de los circuitos integrados y otros componentes, es necesario hacer orificios de diferentes diámetros. La broca que realiza los orificios se debe cambiar cuando todos los orificios de un diámetro específico han sido terminados. El problema se puede ver como una serie de PAV donde las ciudades son los puntos a ser taladrados y la distancia es el tiempo necesario para moverse de una posición a otra [2,3].

- Medición de rayos x. Un aparato especial mide la intensidad de la radiación que se refleja en diferentes posiciones. Para obtener una medición precisa, las lecturas deben de superar 30,000 posiciones. Con base en lo anterior, los puntos de lectura representan las ciudades, y el tiempo que se requiere para moverse de un punto a otro es la distancia entre las ciudades [3].

4.8 Heurísticas para el PAV.

Las heurísticas son procedimientos simples que, se supone, ofrecerán una buena solución (aunque no necesariamente la óptima) a problemas difíciles de una manera fácil y rápida[3].

Las heurísticas existentes para el PAV pueden clasificarse en dos grandes grupos:

- Heurísticas de construcción de rutas. Al principio se unen dos ciudades, después se añaden las ciudades restantes una por una, de tal forma que el costo de la ruta se incremente en una cantidad mínima. Existen algunas variaciones de este tipo de heurísticas, *farthest insertion, nearest insertion, cheapest insertion*, dependiendo de cuáles son los dos nodos que se eligen al iniciar, y lo más importante, cuáles son las ciudades que se insertan en cada etapa [16].

- Heurísticas de mejoramiento de rutas (algoritmos de búsqueda local simple). Este tipo de algoritmos se basa en modificaciones simples, operaciones de intercambios y movimientos, a partir de un recorrido. El objetivo de estas metodologías es producir un recorrido para el cual ninguna operación produzca un mejoramiento. Alternativamente, podemos ver lo anterior como un proceso de *búsqueda de vecindad.* Cada ruta tiene una vecindad asociada,

es decir, recorridos que se pueden producir con un simple movimiento, de esta manera, continuamente se lleva a cabo una búsqueda que termina hasta que no existan mejores vecindades. Entre los algoritmos de búsqueda local simple, los más conocidos son: *el 2-Opt y el 3-Opt* [16].

4.9 Resumen

Durante el desarrollo de este capítulo se presentó de manera general el concepto del PAV, además se describió la formulación matemática del mismo y se utilizó la teoría de las gráficas como herramienta conceptual para presentar una definición formal del problema. Una breve reseña histórica del problema fue presentada, así como, la clasificación en la cual se enmarca al PAV debido a su complejidad. En seguida se expusieron algunas de las formulaciones y aplicaciones prácticas más comunes que se relacionan con el PAV. Para terminar el capítulo, se explicó la forma en que se pueden clasificar las heurísticas existentes para el PAV.

CAPÍTULO 5

AG y PAV

5.1 Introducción

En el presente capítulo se describen las diferentes técnicas que pueden ser aplicadas a cada uno de los procesos que conforman el AG para resolver el problema del agente viajero. Se expone cómo se puede implementar un AG a partir del modelo básico, además se explica una forma alternativa de cambiar su estructura básica introduciendo el concepto de algoritmos genéticos con edades.

5.2 Codificación

En los trabajos de Holland y en los realizados por varios de sus seguidores, los cromosomas son representados por cadenas binarias [5]. Sin embargo, en el contexto del PAV esta codificación no es tan natural, por ejemplo, para un problema de cinco ciudades y 20 aristas (Figura 9), un cromosoma con codificación binaria (Figura 10) representa la ruta que utiliza las aristas 2, 9, 10, 12, y 15 para conectar las cinco ciudades (Figura 11).

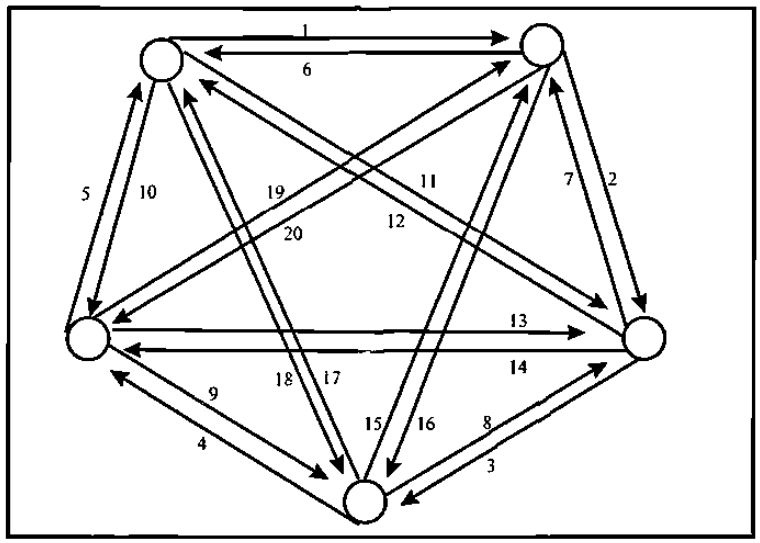


Figura 9. Problema de 5 ciudades y 20 aristas

0	1	0	0	0	0	0	0	1	1	0	1	0	0	1	0	0	0	0	0

Figura 10. Cromosoma con codificación binaria

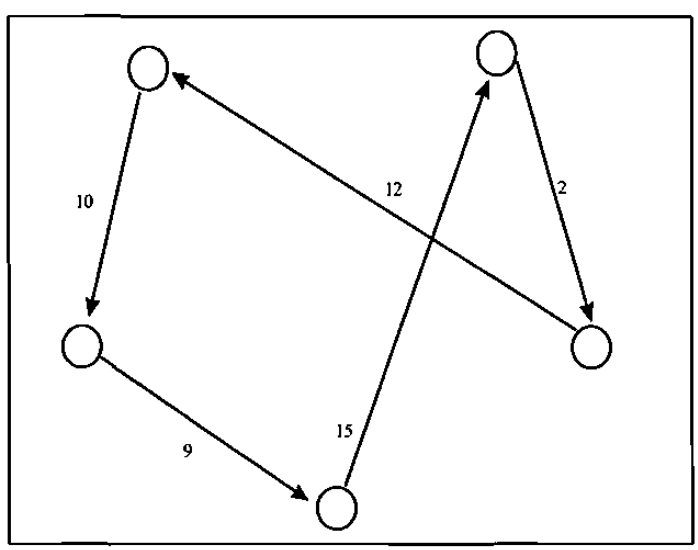


Figura 11. Ruta que representa el cromosoma de la figura 11

Una representación de este tipo no es la más adecuada para el PAV, ya que existe una gran cantidad de cromosomas con cinco "1" y quince "0" que no representan soluciones factibles para el PAV por no formar un circuito válido, por ejemplo: el cromosoma de la figura 12 representa el recorrido invalido de la figura 13.

1	1	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0

Figura 12. Cromosoma de un recorrido invalido

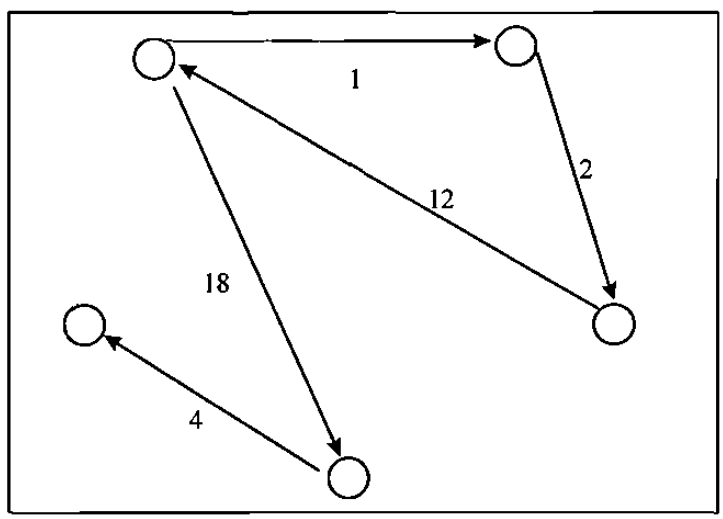


Figura 13. Ruta que representa el cromosoma de la figura 12

Otra forma de representar los recorridos en forma binaria para un PAV de n ciudades es la siguiente:

- cada ciudad se puede codificar como una cadena de $\lceil \log_2 n \rceil$ bits.

- un cromosoma es una cadena de $n*\lceil \log_2 n \rceil$ bits.

Si llevamos a cabo una operación de mutación para este tipo de codificación, puede dar como resultado una secuencia de ciudades que no sea un recorrido válido. Por ejemplo, para un problema de veinte ciudades, donde necesitamos 5 bits para representar una ciudad:

- podemos obtener la misma ciudad dos veces en un cromosoma.

- algunas series de 5 bits (por ejemplo, 10101) no corresponden a alguna ciudad.

Problemas similares se presentan cuando aplicamos la operación de cruzamiento. Una solución que se utiliza cuando se producen recorridos no válidos, es desarrollar algoritmos que rectifiquen los cromosomas (algoritmos reparadores), es decir, que los muevan hacia adentro del espacio-búsqueda [2].

Es un hecho que no existe una forma práctica de codificar los cromosomas mediante cadenas binarias de bits de tal modo que los operadores genéticos clásicos sean eficientes en la resolución del PAV.

Como menciona Michalewics [2]: parece ser que la representación de vector entero es mejor que utilizar algoritmos reparadores. Se puede incorporar dentro de las operaciones cierto conocimiento del problema, de tal forma que se pueda evitar inteligentemente la creación de individuos no factibles. En esta forma particular de abordar el problema, se admite una representación por medio de enteros: un vector $v = \langle i_1, i_2, \ldots, i_n \rangle$ representa un recorrido: desde i_1 a i_2, de i_2 a i_3...de i_{n-1} a i_n y regresa a i_1 (v es una permutación de *(*1 2...*n)*.

Durante el transcurso de los últimos años se han utilizado tres representaciones en forma de vector para trabajar con el PAV: codificación de adyacencia, codificación ordinal y codificación de ruta [2,7]. En el presente estudio sólo trataremos con la codificación de ruta.

La representación en forma de ruta es la manera más natural de codificar las soluciones del PAV, ésta consiste en enumerar las ciudades por orden de recorrido. Por ejemplo, para un problema con nueve ciudades el recorrido

$$5 \rightarrow 1 \rightarrow 7 \rightarrow 8 \rightarrow 9 \rightarrow 4 \rightarrow 6 \rightarrow 2 \rightarrow 3 \hookleftarrow$$

representa el individuo 5, 1, 7 8, 9, 4, 6, 2, 3. Nótese que el individuo 8, 9, 4, 6, 3, 5, 1, 7 también representa al mismo recorrido, sólo que comenzando por la ciudad 8; lo mismo ocurre con el individuo 3, 2, 6, 4, 9, 8, 7, 1, 5, en el cual solamente se cambia el sentido del recorrido.

El problema de esta codificación radica en que el cruce externo clásico entre genes, casi nunca proporciona individuos factibles (Figura 14). Por este motivo se debe recurrir a otros operadores (especiales) de cruzamiento, los cuales se mencionarán en el desarrollo de este capítulo.

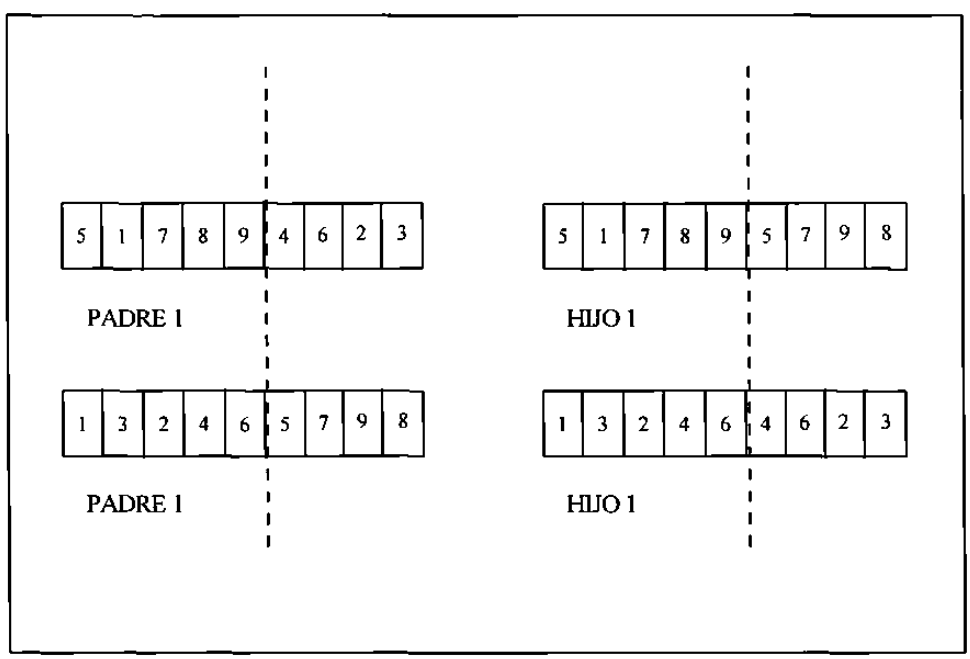


Figura 14. Cruce externo clásico de un punto

5.3 La Población Inicial

La población inicial de un AG puede ser creada de diferentes maneras:

- Podemos aceptar algunas salidas de un algoritmo goloso para el PAV, empezando en diferentes ciudades o,
- Podemos inicializar la población con una muestra aleatoria de las permutaciones de $(1, 2,...,n)$.

5.4 La Aptitud

La evaluación de un cromosoma es directa: para cualquier solución potencial (una permutación de las ciudades), se consulta la tabla con las distancias entre todas las ciudades y, después de $n-1$ operaciones de suma, se puede obtener la longitud total del recorrido [2].

5.5 La Selección

La selección debe dirigir el proceso de búsqueda en favor de los individuos de más alta aptitud. Como se mencionó en el capítulo anterior, existen mecanismos de selección muy variados. El criterio para elegir un mecanismo de muestreo depende del problema y del buen juicio del programador. Para el PAV en el presente estudio se utilizaran los métodos de selección estocástico universal y por torneos.

5.6 Los operadores de cruzamiento

Hasta hace poco, tres operadores de cruzamiento fueron definidos para la representación de ruta [2,4,7]: cruce por emparejamiento parcial (PMX), cruce por orden (OX), y cruce por ciclos (CX) En seguida se explica el operador de cruzamiento por ciclos OX.

OX: Produce la descendencia eligiendo una subsecuencia de un recorrido a partir de un padre y preservando el orden relativo de las ciudades del otro padre. Por ejemplo, dos progenitores (con dos puntos de cruce marcados por el símbolo "|")

p_1 = (1 2 3 | 4 5 6 7 | 8 9) y
p_2 = (4 5 2 | 1 8 7 6 | 9 3)

generan la descendencia de la siguiente manera. Primero, los segmentos que se encuentran entre los puntos de cruce son copiados dentro de la descendencia:

o_1 = (x x x | 4 5 6 7 | x x) y
o_2 = (x x x | 1 8 7 6 | x x).

Los símbolos x se interpretan como "no especificado". Siguiente, comenzando del segundo punto de cruce de un progenitor, las ciudades del otro progenitor son copiadas en el mismo orden, omitiendo las que ya estén presentes. Al llegar al final de la cadena, se continua por el principio de la misma. La secuencia de las ciudades en el segundo progenitor (desde el segundo punto de cruce) es

9-3-4-5-2-1-8-7-6;

después de remover las ciudades 4, 5, 6, y 7 , las cuales ya estaban presentes en el primer descendiente, se obtiene

9-3-2-1-8.

Esta secuencia es colocada en el primer descendiente (comenzando a partir del segundo punto de cruce):

o_1 = (2 1 8 | 4 5 6 7 | 9 3).

Siguiendo los pasos anteriores se puede obtener el otro descendiente:

o_2 = (3 4 5 | 1 8 7 6 | 9 2).

El operador OX explota una propiedad de la representación de ruta, la cual nos dice que lo importante es el orden de las ciudades y no sus posiciones.

5.7 Operadores unitarios

5.7.1 Operador de inversión

Es interesante hacer notar que el operador de inversión sí produce individuos factibles bajo el tipo de representación de ruta. Como se mencionó en el capítulo 2, la inversión simple selecciona dos puntos a lo largo de un cromosoma e invierte el orden de los genes intermedios. Por ejemplo, si aplicamos el operador de inversión a una ruta representada por el cromosoma

(1 2 | 3 4 5 6 | 7 8 9),

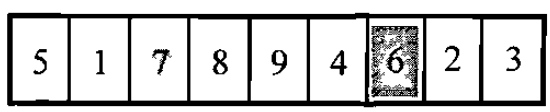
con dos puntos de cruce marcados por el símbolo "|", ésta será transformada en la siguiente ruta:

(1 2 | 6 5 4 3 | 7 8 9).

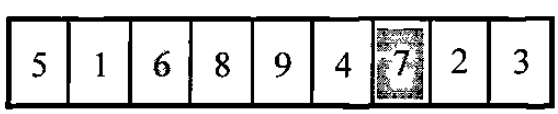
La inversión es un operador unitario, este operador por sí solo no tiene el mismo desempeño que los operadores de recombinación (CX, OX, PMX), por lo que debe complementarse con alguno de ellos [2]. Algunas variaciones del operador de inversión fueron investigadas por David E. Goldberg en 1989 [4].

5.7.2 El operador de mutación

La mutación de un cromosoma que representa un recorrido, puede llevarse a cabo intercambiando dos ciudades que se eligen aleatoriamente. Por ejemplo, el cromosoma



representa una ruta donde los genes (ciudades) con sombra son los que se van a intercambiar. En este caso el cromosoma que se produce es el siguiente:



5.8 AG con edades.

El conocimiento acerca de cuáles podrían ser los valores apropiados para los parámetros de los AG está todavía muy dividido. Entre estos parámetros, el tamaño de la población es el más importante ya que tiene un influencia muy fuerte en el costo

computacional a la hora de ejecutar un AG [2]. La idea principal de variar el tamaño de la población por medio de los AG con edades es la siguiente: diferentes tamaños de la población pueden ser óptimos en diferentes etapas (generaciones) del proceso de búsqueda.

Las perspectivas de los AG con edades son excelentes; aparte del ya comentado control automático del tamaño de la población, parecen proporcionar mecanismos de selección más robustos que los ya conocidos.

5.9 Autocontrol del crecimiento excesivo de la población

Como menciona Michalewicz en su libro publicado en 1994 [2], la estrategia lineal es la que presenta mejores resultados en comparaciones con las otras, desafortunadamente también es la que presenta el costo computacional más alto. Atendiendo a lo anterior, en esta tesis, se propone la introducción de la siguiente idea: se ha observado que algunas especies naturales, entre ellas los pingüinos, tienen un mecanismo, si se le puede llamar así, de autocontrol en el tamaño de su población. Cuando se presenta una sobrepoblación en algún grupo de esta especie, parte de la misma es sacrificada mediante un suicidio colectivo. Esta observación no es la única, en la naturaleza podemos encontrar cómo algunas especies autoregulan su población cuando ésta ha crecido en demasía y como consecuencia provoca la escasez de sus recursos (alimentación).

Con base en las observaciones anteriores, podemos desarrollar un algoritmo que autoregule el tamaño de la población cuando ésta ha excedido los límites de consumo de sus recursos (en este caso la memoria del computador). Para llevar a cabo esta autoregulación se simulan torneos entre toda la población hasta completar una cantidad específica de supervivientes. Esta idea se implementará en el siguiente capítulo y se

presentarán, por primera vez, algunos resultados que se obtuvieron con pruebas realizadas, utilizando este enfoque.

5.10 Resumen

En este capítulo se mostraron algunas técnicas que se pueden utilizar en los diferentes procesos que constituyen a un AG al momento de atacar el PAV. Además, se presentó un nuevo mecanismo para controlar el crecimiento desmedido en la población, que se produce al utilizar la estrategia lineal de asignación de edades en los individuos.

CAPÍTULO 6

PRUEBAS Y RESULTADOS EXPERIMENTALES DEL ESTUDIO

6.1 Introducción

Durante el desarrollo de este capítulo se explicará la forma de utilizar los dos sistemas que se emplearon para obtener los resultados finales y las pruebas que se realizaron. Este capítulo se dividió en dos partes: la primera utiliza un enfoque basado en un AG normal, el cual nombraremos AG estándar, y la segunda parte utiliza el paradigma de los algoritmos genéticos con edades (AGE).

6.2 AG estándar

Las principales características del AG estándar son las siguientes:

1. Se utiliza la representación en forma de ruta.

2. La población inicial se puede producir de dos maneras: se inicializa a los individuos aleatoriamente o se utiliza la salida de un algoritmo goloso.

3. El método de selección puede ser el universal ó el de torneos.

4. Los operadores de cruzamiento que se pueden utilizar son los siguientes OX, PMX, CX.

5. La mutación se lleva a cabo intercambiando dos ciudades de un cromosoma al azar.

6.2.1 Utilización del sistema del AG estándar

Para llevar a cabo las pruebas, se elaboró un sistema de computadora en lenguaje C (el cual se incluye en un disco flexible de computadora). Mediante este programa se pueden capturar los parámetros que utiliza el AG estándar para su inicialización y ejecución (figura 15).

```
C:\>AG
------- Entrada de datos e inicialización para el AG -------
Nombre del Archivo de salida: prueba1                                  prueba1
Tamaño de la población inicial                                         100
Longitud del cromosoma                                                 100
Imprimir los cromosomas ? (s/n)                                        s
Generar la población al azar ? (s/n)                                   s
Máximo numero de generaciones                                          500
Probabilidad de cruzamiento                                            0.78
Probabilidad de mutación                                               0.015
Imprimir cada cuántas generaciones?                                    10
Semilla Inicial para el generador de números aleatorios , de 0.0 a 1.0 0.31
Nombre del archivo de distancias: 100.bin                              100.bin
```

Figura 15. Entrada de parámetros del AG estándar

Cuando la ejecución del programa llega a su fin, los resultados que genera son almacenados en un archivo con el nombre que se proporcionó en la entrada de datos e inicialización para el AG. A continuación se muestra la salida que genera el sistema:

```
Método de selección                = Por torneo
Tamaño del torneo                  = 5
Tamaño de la población             = 90
Longitud del cromosoma             = 100
Máximo número de generaciones      = 10000
Probabilidad de cruzamiento        = 0.780000
Probabilidad de mutación           = 0.015000
Número de cruzamientos             = 693482
Número de mutaciones               = 26722
Mejor Aptitud Global               = 22970.198470
Encontrada en la generación        = 7206

15 93 21 69 65 64 25 3 96 55 79 30 88 41 7 91 0 62 5 48 74 18 89 46 92 27 57
66 63 39 53 1 43 67 84 38 29 95 77 51 4 36 32 75 12 94 81 49 72 68 80 24 60 50
```

86 8 6 56 19 11 54 82 28 70 40 99 47 13 2 42 45 33 26 85 34 61 59 76 22 97 90 44 31 10 14 16 58 73 20 71 9 83 35 98 37 23 17 78 52 87

Peor Aptitud Global = 58232.425782
Encontrada en la generación = 1057

55 79 30 88 41 7 91 0 62 5 48 89 18 4 25 87 98 46 95 77 51 74 36 32 75 92 27 57 66 63 39 53 1 43 49 72 20 71 9 83 35 37 23 17 78 52 15 21 93 69 65 33 3 96 67 84 38 29 12 94 81 68 80 24 60 50 86 8 54 82 28 45 2 42 13 70 40 99 47 64 6 56 19 11 26 85 34 61 59 76 22 97 90 44 31 10 14 16 73 58

6.2.2 Resultados obtenidos mediante el AG estándar

Para realizar las pruebas se utilizó una instancia en particular del problema, que se define como un conjunto de ciudades sobre un espacio planar. Este problema fue planteado por Krolak, Felts y Nelson, y su costo óptimo, el cual se conoce, tiene un valor de 21282 [14]. En la tabla I se especifican las coordenadas de cada una de las 100 ciudades del problema.

Tabla I. Problema de 100 ciudades de Krolak, Felts y Nelson

1	1380	939	26	178	24	51	2482	1183	76	3893	102
2	2848	96	27	2678	1825	52	3854	923	77	2178	1619
3	3510	1671	28	1795	962	53	376	825	78	3822	899
4	457	334	29	3384	1498	54	2519	135	79	378	1048
5	3888	666	30	3520	1079	55	2945	1622	80	1178	100
6	984	965	31	1256	61	56	953	268	81	2599	901
7	2721	1482	32	1424	1728	57	2628	1479	82	3416	143
8	1286	525	33	3913	192	58	2097	981	83	2961	1605
9	2716	1432	34	3085	1528	59	890	1846	84	611	1384
10	738	1325	35	2573	1969	60	2139	1806	85	3113	885
11	1251	1832	36	463	1670	61	2421	1007	86	2597	1830
12	2728	1698	37	3875	598	62	2290	1810	87	2586	1286
13	3815	169	38	298	1513	63	1115	1052	88	161	906
14	3683	1533	39	3479	821	64	2588	302	89	1429	134
15	1247	1945	40	2542	236	65	327	265	90	742	1025
16	123	862	41	3955	1743	66	241	341	91	1625	1651
17	1234	1946	42	1323	280	67	1917	687	92	1187	706
18	252	1240	43	3447	1830	68	2991	792	93	1787	1009
19	611	673	44	2936	337	69	2573	599	94	22	987
20	2576	1676	45	1621	1830	70	19	674	95	3640	43
21	928	1700	46	3373	1646	71	3911	1673	96	3756	882
22	53	857	47	1393	1368	72	872	1559	97	776	392
23	1807	1711	48	3874	1318	73	2863	558	98	1724	1642
24	274	1420	49	938	955	74	929	1766	99	198	1810
25	2574	946	50	3022	474	75	839	620	100	3950	1558

La población inicial fue creada de una manera aleatoria. El número de generaciones que se utilizó fue de 10,000 y los parámetros son los mismos que se mostraron en la figura 15. Además, se experimentó utilizando como población inicial la salida de un algoritmo goloso. Los resultados obtenidos se muestran en la tabla II.

Tabla II. Longitudes de los recorridos que se obtuvieron mediante el AG estándar

	Población inicial aleatoria		Población inicial utilizando un algoritmo goloso
Operador de cruce	Método de selección		Método de selección
	Universal	Por torneos	Por torneos
OX	54599	23935	22508
CX	51602	47282	24670
PMX	47301	45966	24308

6.2.3 Sumario del AG estándar

Los mejores resultados se obtuvieron mediante el OX; sin embargo, esto no se debe interpretar como un hecho concluyente, pues no se ha llevado a cabo un estudio experimental exhaustivo del comportamiento de los operadores. Por otra parte, pensamos que la utilización de un algoritmo goloso para generar la población inicial es una forma útil de obtener mejores resultados. Sin embargo, no presta ninguna ayuda al momento de probar el desempeño mismo del AG, por lo que prescindiremos de él a la hora de probar el AGE en la segunda parte de este capítulo. A continuación se presentan algunas gráficas (figuras 16, 17 y 18) con los resultados experimentales.

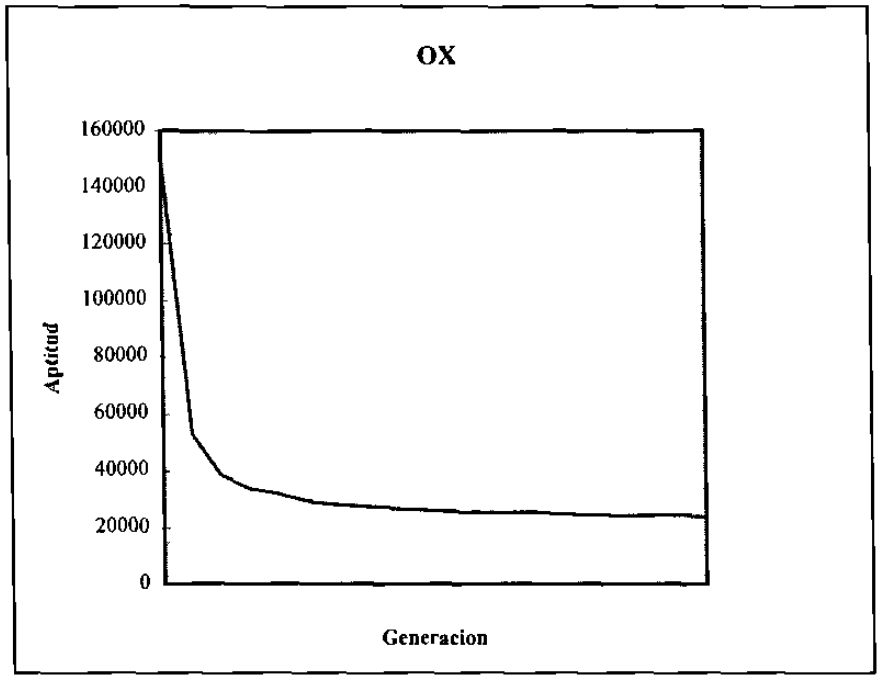


Figura 16. Resultados experimentales de OX con el método de selección por torneos

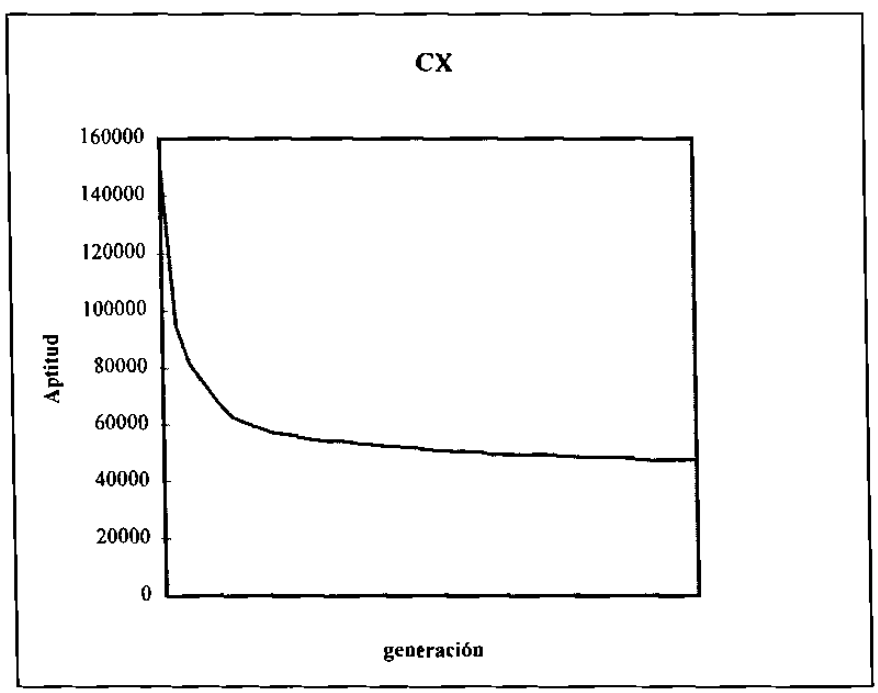


Figura 17. Resultados experimentales de CX con el método de selección por torneos

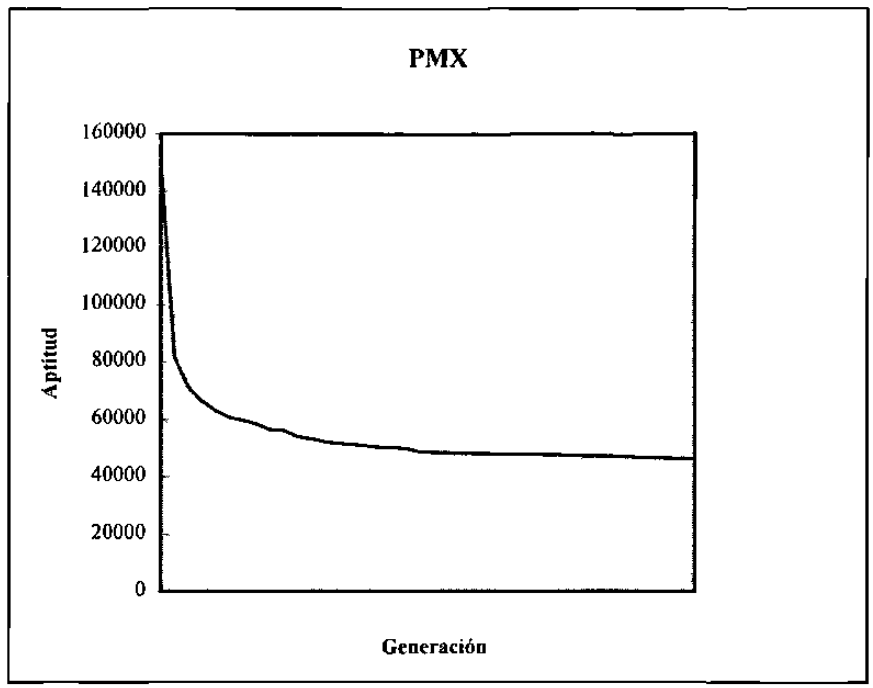


Figura 18. Resultados experimentales de PMX con el método de selección por torneos

6.3 AGE

Las principales características del AGE son las siguientes:

1. Se utiliza la representación en forma de ruta.

2. La población inicial se genera de la siguiente manera: se inicializa a los individuos aleatoriamente (como se mencionó en el apartado anterior, no se utilizará el algoritmo goloso para generar la población al azar).

3. Se descarta por completo el método de selección. En lugar de ello se asigna una duración (tiempo de vida) a cada uno de los cromosomas.

4. La estrategia que se utiliza para asignar las edades a los individuos es la de asignación lineal.

5. Es importante hacer notar que en este punto es donde se implementa el nuevo mecanismo que regula el crecimiento desmedido de la población, para ello se establece un límite de tamaño en la población. Cuando éste ha sido alcanzado se simula una competencia general, con la cual se obliga a que disminuya el tamaño de la misma a una cantidad de supervivientes dada. La cantidad de supervivientes se puede definir dentro del código del programa. La forma de hacerlo es muy sencilla, ya que solamente se especifica en un #define (ver Apéndice). Con esto se pretende, como se mencionó en el capítulo 5, autoregular el tamaño de la población, evitando de esta manera el crecimiento desmedido que provoca la estrategia de asignación lineal.

6. Los operadores de cruzamiento que se pueden utilizar son los siguientes OX, PMX, CX.

7. La mutación se lleva a cabo intercambiando dos ciudades de un cromosoma al azar.

6.3.1 Utilización del sistema AGE

Para llevar acabo las pruebas, también se elaboró un sistema de computadora en lenguaje C (ver apéndice). La manera de operar este programa es similar a la del AG estándar, la diferencia es que el sistema pide un parámetro adicional: la máxima edad permitida. En la figura 19 se muestra la manera de iniciar el AGE.

Cuando la ejecución del programa llega su fin, los resultados que genera se almacenan en un archivo igual que en el AG estándar.

```
C:\>AGE
------- Entrada de datos e inicialización para SGA ------
Nombre del Archivo de salida:                                   prueba2
Tamaño de la población inicial                                  10
Longitud del cromosoma                                          100
Generar la población al azar ?                                  s
Imprimir los cromosomas ? (s/n)                                 s
Máximo número de generaciones                                   10000
Probabilidad de cruzamiento                                     0.78
Probabilidad de mutación                                        0.015
Máxima edad permitida                                           5
Imprimir cada cuántas generaciones?                             500
Semilla Inicial para el generador de números aleatorios , de 0.0 a 1.0  0.31
Nombre del archivo de distancias:                               100.bin
```

Figura 19. Entrada de parámetros del AGE

La salida que genera el sistema, cuando la ejecución del programa llega su fin, es similar a la que se presenta en la primera parte de este capítulo, con la diferencia de que se muestran los parámetros adicionales de: número de supervivientes y máxima edad permitida.

6.3.2 Resultados obtenidos mediante el AGE

Para realizar las pruebas se utilizó la misma instancia del problema anterior (tabla I). La población inicial fue creada de una manera aleatoria. El número de generaciones que se utilizó fue de 10,000 y los parámetros son los mismos que se mostraron en la figura 19, excepto la probabilidad de mutación, la cual se cambio a 0.2 para los casos en

que se utilizaron los operadores de cruzamiento CX y PMX. Los resultados obtenidos se muestran en la tabla III

Tabla III. Longitudes de los recorridos que se obtuvieron mediante el AGE

Algoritmos genéticos con edades	
Operador de cruce	Población inicial aleatoria
OX	23558
CX	37089
PMX	35367

6.3.3 Sumario del AGE

Los mejores resultados se obtuvieron mediante el operador OX. Sin embargo, como se expresó anteriormente, esto no se debe interpretar como un hecho concluyente. Como punto final de este capítulo se muestran algunas gráficas del comportamiento del AGE (figuras 20, 21, 22). Observe cómo varía el tamaño de la población a través del paso de las generaciones y cómo la aptitud mejora, también, cuando avanza el número de generaciones.

6.4 Resumen

Durante el desarrollo de este capítulo se presentaron algunos resultados experimentales adoptando dos enfoque diferentes; el primero hace uso de dos métodos de selección ampliamente utilizados, mientras que el segundo utiliza el paradigma de los AGE y además incorpora una manera de controlar el tamaño desmedido de la población,

que se genera al utilizar la estrategia de asignación de edad lineal. En el siguiente capítulo se presentarán algunas recomendaciones para realizar investigaciones futuras relacionadas con el tema tratado en esta segunda parte del capítulo.

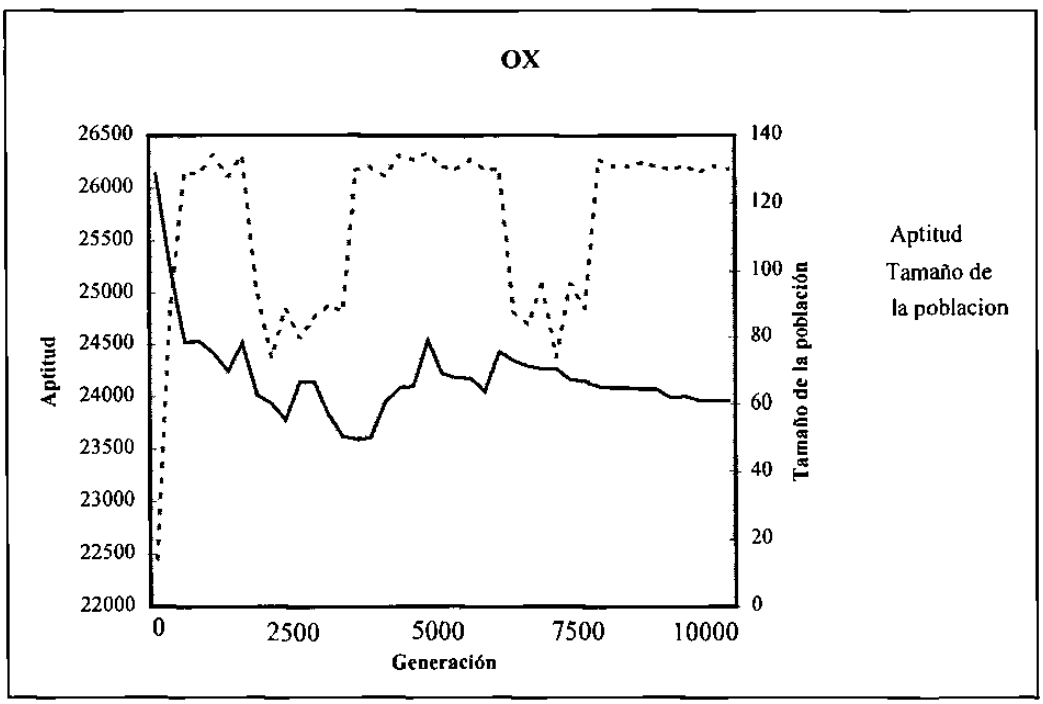


Figura 20. Resultados experimentales de OX del AGE

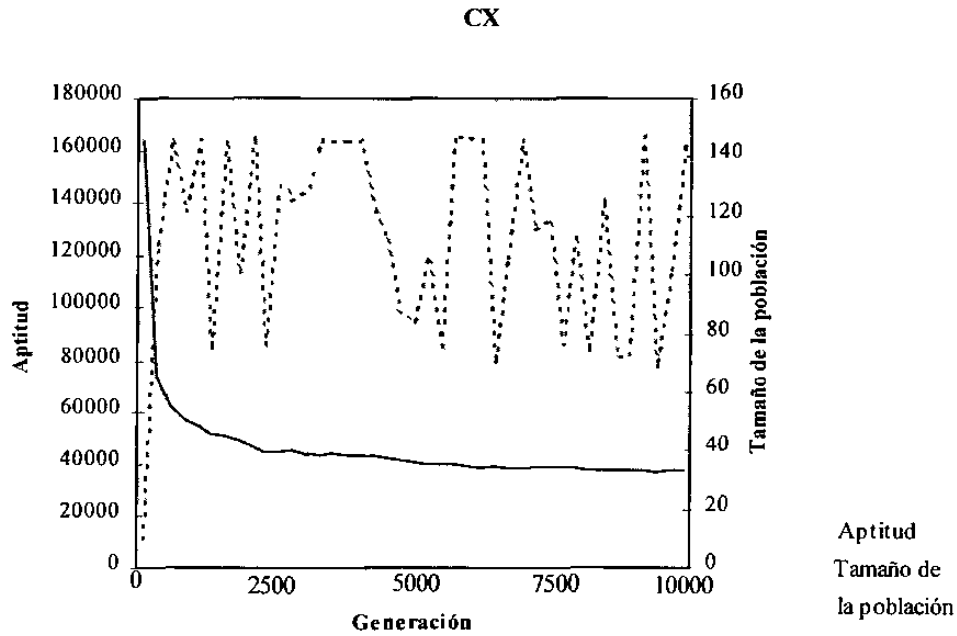


Figura 21. Resultados experimentales de CX del AGE

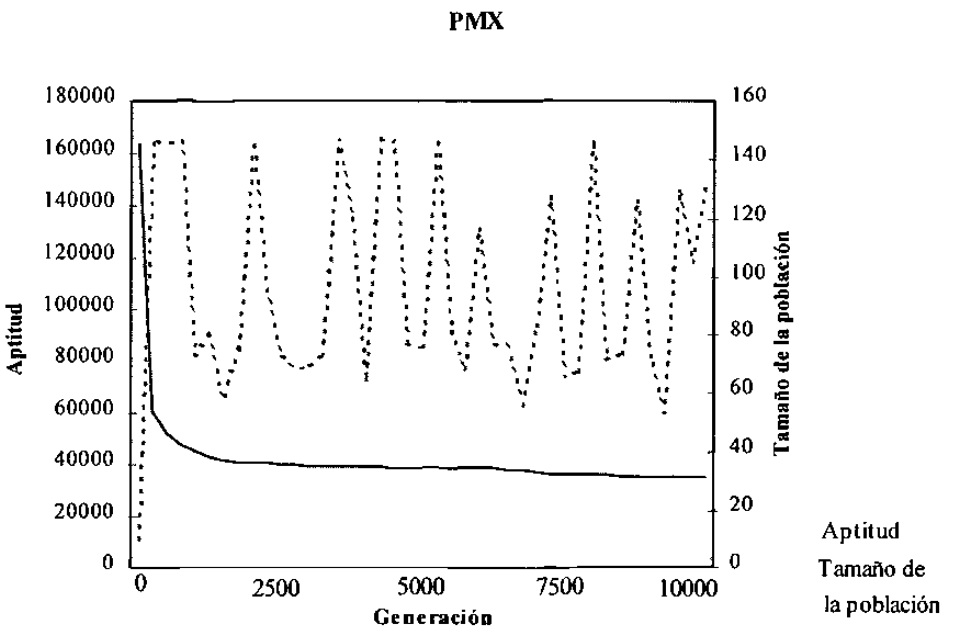


Figura 22. Resultados experimentales de PMX del AGE

CAPÍTULO 7

CONCLUSIONES Y RECOMENDACIONES

7.1 Conclusiones generales

A lo largo de este trabajo se han abordado tres temas importantes: El primero de ellos fue el de presentar un visión general de los AG, su origen, su composición y la manera de llevar a cabo implementaciones del mismo. El segundo consistió en exponer de una manera sencilla y concisa el ya conocido y famoso problema del agente viajero y de mostrar cómo se puede utilizar la metodología de los AG para abordar este problema. El tercero, que constituyó el objetivo fundamental de la tesis, fue mostrar cómo adoptando ideas aparentemente muy sencillas, en este caso la introducción de los conceptos de edad y autocontrol de la población en los AG, se pueden abordar algunos problemas de una mejor manera. Algo que podemos aprender de lo anterior es que nada está completamente acabado y que ahí, donde aparentemente ninguna cosa hay por

hacer, existe alguna manera de introducir pequeños cambios que nos pueden llevar a generar u obtener nuevos resultados.

La utilización del concepto de edad en los AG no es completamente nueva, sin embargo, no se había experimentado en el PAV. Los resultados obtenidos en este estudio muestran como el AGE produce mejores resultados en comparación con el AG estándar en el caso donde la población inicial es generada de manera aleatoria.

Por otra parte, como se mencionó en el capítulo anterior los mejores resultados se obtuvieron utilizando el operador de cruzamiento OX. Sin embargo, cabe recalcar, que esto no se debe tomar como un hecho concluyente, pues para ello se tendría que llevar acabo un estudio experimental exhaustivo sobre el comportamiento de los operadores especiales de cruzamiento para el PAV.

El AGE produce buenos resultados sin un costo computacional muy grande, por lo que se concluye que la utilización del mismo en el caso del PAV debe aprovecharse para generar nuevas investigaciones.

Concluimos este trabajo de tesis exhortando a las nuevas generaciones de investigadores para que alimenten la voluntad de indagar y experimentar. Las prestaciones que se obtienen utilizando este paradigma son alentadoras, además es una nueva opción en las investigaciones que giran en torno del PAV. A continuación se presentan algunas investigaciones que podrían realizarse.

7.2 Recomendaciones futuras

Como se mencionó en el capítulo 5, las perspectivas de los AG con edades son excelentes. En el momento presente esta línea de investigación tiene tres frentes abiertos: estudio de sus prestaciones en una variedad de problemas lo más amplia posible,

encontrar mecanismos eficientes en la asignación de las edades a los individuos y un estudio teórico de la convergencia.

Por otra parte, para el caso del PAV, se pueden realizar estudios sobre el comportamiento de los operadores especiales de cruzamiento (CX, OX, y PMX), y desarrollar algoritmos híbridos, mediante la mezcla del AGE y otras heuristicas conocidas para el PAV.

REFERENCIAS

[1] Adenzo diaz, Fred Glover, Hassan M. Ghaziri, J.L. Gonzalez, Manuel Laguna Pablo Moscato, Fan T.Tseng. "Optimización Heurística y redes neuronales". Editorial Paraninfo, 1996.

[2] Zbigniew Michalewicz. "Genetic Agorithms + Data Structures = Evolution Programs". Springer-Verlag, 1994.

[3] Mark H. Noschang. "The Traveling Salesman Problem - a Review of Theory and Current Research -". University of Cincinnati, 1996.

[4] David E. Goldberg. "Genetic Algorithms in Search Optimization & Machine Learning". Addison-Wesley Co., Inc, Reading, MA, 1989.

[5] Melanie Mitchell. "An Introduction to Genetic Algorithms". The MIT Press. Cambridge, Massachusetts. London, Englan, 1996.

[6] David Baesley., David R. Bull., Ralph R. Martin. "An Overview of Genetic Algorithms: Part 1, Fundamentals". University Computing, 15(2) 59-69, 1993.

[7] Anselmo Pérez Serrada "Una Introducción a la Computación Evolutiva". Anselmo@hp9000.cpd.uva.es.,1996.

[8] Hsiao-Lan Fang "Investigating Genetic Algorithms for Scheduling". MSc Dissertation. Depaertament of Artificial Intelligence. University of Edinburgh, 1992.

[9] Emma Collingwood. "Investigation of Multiple Chromosome Evolutionary Algorithm for Bus Scheduling and Other Problems". MSc Information Technology: Knowledge Based Systems. Depaertament of Artificial Intelligence. University of Edinburgh, 1995.

[10] Paul Field. "A Multary Theory for Genetic Algorithms: Unifying Binary and Nonbinary Problem Representations". Submitted to the University of London for the Degree of Doctor of Philosophy in Computer Science. Ftp.dcs.qmw.ac.uk/applied-logic/pg/field/MultaryTheory.ps.Z., 1996.

[11] Brad L. Miller. And David E. Goldberg. "Genetic algorithms, selection schemes, and the varying Effects of noise". IlliGAl Report No. 95009, 1995.

[12] David Baesley, David R. Bull, Ralph R. Martin. "An Overview of GA: part 2, Research Topics ". University Computing, 15(2) 59-69, 1993.

[13] Michael R. Gary and David S. Johnson. "Computers and Intractability", 1978.

[14] Ordoñez Reinoso Ivan. "Soluciones al Problema del Vendedor Viajero Generalizado mediante AG con Operadores Especiales de Cruce". Tesis en Opción al Grado de Maestría en Ciencias con Especialidad en Ciencias Computacionales. ITESM Monterrey, 1991.

[15] Reimar Hofmann. "Examinations on the algebra of Genetic Algorithms". Diploma Thesis. Tecnical University of Computer Science, 1993.

[16] David S. Johnson, Lyle A. McGeoch. The Traveling Salesman Problem: A Case Study in Local Optimization. www.research.att.com/~dsj papers.html, 1997.

APÉNDICE

CÓDIGO FUENTE DEL AGE

APÉNDICE

CÓDIGO FUENTE DEL AGE

El siguiente programa contiene todas las funciones necesarias para ejecutar el AGE. Al final de la tesis se anexa un disco flexible con el archivo ejecutable del mismo.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <conio.h>
#include <time.h>
#include<math.h>
#include <malloc.h>
#include "sga.h"

/* DECLARACION DE PROTOTIPOS*/
int     aleat();
double  apto(int);
int     borrar(double);
int     busca(double);
void    distancias();
void    generation();
void    inicializa();
void    initpop();
void    insertar(double);
void    initdata();
void    lee(void);
void    lee instancia();
void    limpia(int sale);
void    mutacion(int);
void    nombre archivo();
double  objetivo();
void    pantalla();
void    parametros();
void    poblacion(int j);
void    pop_azar();
void    preselect();
void    recorre(int i);
int     regresa menor(int desde1);
void    reporte_inicial();
double  ruta_menor(void);
void    salida mejor();
void    sal mejor_gen();
void    supervivientes(void);
static void reset();
void    revuelve();
 * DECLARACION DE VARIABLES GLOBALES PARA LA POBLACION INICIAL */
int     numcities,recorrido[maxcity];
```

```
double  distbetween[maxcity][maxcity],  distancia[maxcity][maxcity],v,w;
int     cantidad;
int     sal  gen–0;
int     mul=0;
static  FILE *fp  texto   NULL;
int randomlst[maxcity],intarraybak1[maxcity],intarraybak2[maxcity],intarray2[maxcity];
int tmparray[maxcity],
char which  crossover[] = "ox.c";


/* VARIABLE GLOBALES */
static int tourneylist[MAXPOP], tourneypos;
static int firsttime = 1;

/*-------------------------------------------------------------*/
/*              PROGRAMA PRINCIPAL             */
/*-------------------------------------------------------------*/
main()
{
  ncross=0;
  nmutation  0;
  infp – stdin;
  outfp = stdout;
  system("cls");
  inicializa(); /* Inicializa los parámetros */
  calcula_edad();
  report();
  for(gen=1; gen<=maxgen; gen++){
   generation();/* Inicia una nueva generación */
   calcula  edad();
   borrar((max_edad-1));
   report();
   /* Avanza la generación */
   }
   pantalla();
   parametros();
 fclose(fptext);
}

/*-------------------------------------------------------------------------------*/
/*    ENTRADA DE DATOS PARA LA INICIALIZACION DEL AG          */
/*-------------------------------------------------------------------------------*/
void inicializa()
{
  initdata();/* Obtiene los principales parámetros del algoritmo *
  randomize();
   * inicializa valores y contadores globales *
        nmutation – 0;
  ncross – 0;
  bestfit[0].fitness – 0.0;
  bestfit[1].fitness – 0.0;
  bestfit[0].generation – 0;
  bestfit[1].generation – 0;
  initpop();
```

```
  statistics();
}

/*-----------------------------------------------------------------*/
/*                PARAMETROS DE LA FUNCION                         */
/*-----------------------------------------------------------------*/
void initdata()
{
   char answer[2];
   char resp[2];
   system("cls");
   printf("\n ------- Entrada de datos e inicialización para SGA -------\n");
   nombre_archivo();
   printf(" Tamaño de la población inicial ----------> ");
   scanf("%d", &popsize);
   if((popsize%2) != 0) {
         printf("El tamaño de la población debe ser par! \n Incrementando tamaño de la población en
uno.\n");
         popsize++;
    };
    printf(" Longitud del cromosoma----------> ");
    scanf("%d", &lchrom);

    printf(" Generar la población al azar ? (s/n) ----------> ");
    scanf("%s",resp);
          if(strncmp(resp,"n",1) == 0) azar = 0;

    printf(" Imprimir los cromosomas ? (s/n) ----------> ");
    scanf("%s",answer);
          if(strncmp(answer,"n",1) == 0) printstrings = 0;

    printf(" Máximo número de generaciones ----------> ");
    scanf("%d", &maxgen);
    printf(" Probabilidad de cruzamiento ----------> ");
    scanf("%f", &pcross);
    printf(" Probabilidad de mutación ----------> ");
    scanf("%f", &pmutation);
    printf(" Máxima edad permitida ----------> ");
    scanf("%d", &max_edad);
    printf(" Imprimir cada cuantas generaciones?----------> ");
    scanf("%d", &sal_gen);
 }

 /*-----------------------------------------------------------*/
 /*                  GENERACION                               */
 /*-----------------------------------------------------------*/
 void generation()
 {
  int i;
  popsize=(talla-1);
  for(i=0;i<popsize;i++){
   if(talla>=(MAXPOP-4)){
     statistics();
     calcula_edad();
```

```
    report();
    supervivientes();
    break;
   }
   crossover(i);
   mutacion(i);
   mutacion(i+1);
  }
  statistics();
}

/*-------------------------------------------------------------------------*/
/*      FUNCIONES PARA LA POBLACION INICIAL                                */
/*-------------------------------------------------------------------------*/
void initpop()
{
 int  j;
 clrscr();
 lee();
 talla=0;
 if(azar==1) {
  pop  azar();
 }
 else {
 for(j=0;j<popsize;j++)  poblacion(j);
 }
 minabsoluto = lista[0].objetivo;
 maxabsoluto =  minabsoluto;
}

/*-------------------------------------------------------------------------*/
/*      FUNCION QUE OBTIENE LAS DISTANCIAS                                 */
/*-------------------------------------------------------------------------*/
void lee() {
static FILE   *fpread;
char          tablefile[13];
int           i, j;
printf(" Nombre del archivo de distancias: ");
fscanf( stdin,"%13s", tablefile);
if ((fpread = fopen(tablefile, "rb")) == NULL) {
          fprintf(stderr, "\n Error al abrir el archivo de distancias. Si no lo tienes");
          fprintf(stderr, "crea uno ,\n utiliza  visor.exe ");
          fprintf(stderr, "para hacerlo.\n");
          exit(-1);
}
if ((fread(&numcities, sizeof(int), 1, fpread) != 1)) {
          fprintf(stderr, "Error al abrir el archivo.\n");
          exit(-1);
}
 for (i = 0; i <lchrom; i++) {
  for (j = 0; j < lchrom; j++) {
   if (fread(&(distbetween[i][j]), sizeof(double), 1, fpread) != 1) {
    fprintf(stderr, " Error al leer el archivo.\n");
    exit(-1);
```

```
   }
  }
 }
 fclose(fpread);
}


/*------ ----------------------------------------------------------------*/
/*     REGRESA EL NODO MAS CERCANO                                       */
/*-------------------------------------------------------------------------*/
int regresa_menor(int desde1)
{
 int   i,menor;
 double paso;
 menor = 0.0;
 paso = 1000000;
 for(i=0;i<lchrom;i++)
  if((distancia[desde1][i]!=0.0)&&(distancia[desde1][i]<paso)){
            paso=distancia[desde1][i];
            menor=i;
    }
 limpia(menor);
 return menor;
}

/*-------------------------------------------------------------------------*/
/*     REGRESA UN NODO ALEATORIO                                         */
/*-------------------------------------------------------------------------*/
int aleat()
{
 return rnd(1,9);
}

/*-------------------------------------------------------------------------*/
/*     LIMPIA UN NODO Y SUS CONECCIONES                                  */
/*-------------------------------------------------------------------------*/
void limpia(int sale)
{
 int i,j;
 for (j=0;j<lchrom;j++)
   distancia[j][sale]=0.0;
}

/*-------------------------------------------------------------*/
/*     REGRESA UNA RUTA                                        */
/*-------------------------------------------------------------*/
double ruta_menor(void){
  int   i,inicio;
  distancias();
  inicio=aleat();
  recorrido[0] = inicio;
  limpia(inicio);
  for(i=1; i<lchrom;i++){
    recorrido[i]=regresa_menor(inicio);
```

```
   inicio=recorrido[i];
  }
return objetivo();
}


/*-----------------------------------------------------------------*/
/*     CREA UNA MATRIZ DE DISTANCIAS                               */
/*-----------------------------------------------------------------*/
void distancias()
{
 int i,j;
 for (i = 0; i <lchrom; i++)
  for (j = 0; j < lchrom; j++)
   distancia[i][j]=distbetween[i][j];
}

/*-----------------------------------------------------------------------------*/
/*     CREA CROMOSOMAS CON UN ALGORITMO GOLOSO                                 */
/*-----------------------------------------------------------------------------*/
void poblacion(int j)
{
 int   i,k;
 double costo1,costo;
 costo = 1000000.00;
 for(k=0;k<2;k++){
   costo1=ruta_menor();
   if (costo1<costo){
     costo=costo1;
     insertar(costo);
   }
 }
}

/*-----------------------------------------------------------------*/
/*            REGRESA EL COSTO DE UN RECORRIDO                     */
/*-----------------------------------------------------------------*/
double  objetivo()
{
 int   i,a,b,primera,ultima;
 double costo;
 costo=0;
 distancias();
 primera=recorrido[0];
 ultima =recorrido[lchrom-1];
 for (i=0;i<(lchrom-1);i++){
            a=recorrido[i];
            b=recorrido[i+1];
            costo=costo+distbetween[a][b];
  }
 costo = costo+distbetween[ultima][primera];
return  costo;
}
```

```
/*-------------------------------------------------------------------------*/
/*            REGRESA EL COSTO DE UN CROMOSOMA          */
/*-------------------------------------------------------------------------*/
double apto(int actual)
{
 int   i,a,b,primera,ultima;
 double costo;
 costo = 0;
 primera = lista[actual].cromosoma[0];
 ultima  = lista[actual].cromosoma[lchrom-1];
 for (i=0;i<(lchrom-1);i++){
          a=lista[actual].cromosoma[i];
          b=lista[actual].cromosoma[i+1];
          costo=costo+distbetween[a][b];
 }
costo= costo+distbetween[ultima][primera];
return  costo;
}

/*-------------------------------------------------------------------------*/
/*            CALCULA LAS ESTADISTICAS          */
/*-------------------------------------------------------------------------*/
statistics()
{
   int i, j;
   sumfitness = 0.0;
   min = lista[0].objetivo;
   max = lista[0].objetivo;

   /* Encuentra el min max y sumfitness */
   for(j=0;j<talla;j++){
     sumfitness = sumfitness + lista[j].objetivo;  /* Acumulada */
     if(lista[j].objetivo > max){ max =lista[j].objetivo; /* Nuevo max */
        for(i = 0; i < lchrom; i++){
             bestfit[0].chrom[i]  = lista[j].cromosoma[i];
        }
         bestfit[0].fitness   = lista[j].objetivo;
         bestfit[0].generation = gen;
     }
     if(lista[j].objetivo < min){ min = lista[j].objetivo; /* Nuevo min */
           for(i = 0; i < lchrom; i++){
             bestfit[1].chrom[i]  = lista[j].cromosoma[i];
        }
         bestfit[1].fitness   = lista[j].objetivo;
         bestfit[1].generation = gen;
     }
    }
   /* Calcula la aptitud promedio */
   avg = sumfitness/talla;
   /* Registra la aptitud máxima absoluta */
  if (bestfit[0].fitness>maxabsoluto){maxabsoluto=bestfit[0].fitness;
    for(i = 0; i < lchrom; i++)
             aptglobal[0].chrom[i]  = bestfit[0].chrom[i];
```

```
        aptglobal[0].fitness    = bestfit[0].fitness;
        aptglobal[0].generation = gen;
  }
  /* Registra la aptitud máxima absoluta */
  if (bestfit[1].fitness<minabsoluto){minabsoluto=bestfit[1].fitness;
    for(i = 0; i < lchrom; i++)
            aptglobal[1].chrom[i]  = bestfit[1].chrom[i];
        aptglobal[1].fitness   = bestfit[1].fitness;
        aptglobal[1].generation = gen;
  }
}

/*-------------------------------------------------------*/
/*            PROCESO DE MUTACION                        */
/*-------------------------------------------------------*/
void mutacion(int i)
{
 int l,k,temp1,temp2;
 if (!flip(pmutation))
 return;
 nmutation++;
 l = rnd(0,lchrom-1);
 k = rnd(0,lchrom-1);
 lista[i].xmuta[0]=l;
 lista[i].xmuta[1]=k;
 temp1 = lista[i].cromosoma[l];
 temp2 = lista[i].cromosoma[k];
 lista[i].cromosoma[l]=temp2;
 lista[i].cromosoma[k]=temp1;
 lista[i].objetivo=apto(i);
 lista[i].edad= 0.0;
}

/*-------------------------------------------------------------------------------*/
/* CALCULA LA EDAD DE LOS INDIVIDUOS POR ASIGNACION LINEAL */
/*-------------------------------------------------------------------------------*/
calcula_edad()
{
 int i;
 for(i=0;i<talla;i++) {
  if(lista[i].edad ==0.0)
   lista[i].edad =1+((max  edad-1) * ((lista[i].objetivo - minabsoluto) /
      (maxabsoluto - minabsoluto)));
     else {
      edad1=lista[i].edad;
      lista[i].edad=(floor(edad1)+1);
     }
   }
 }

 /*-------------------------------------------------------------------*/
  * INSERTA UN NUEVO ELEMENTO EN LA POBLACION          *
  *-------------------------------------------------------------------*
  void insertar(double dato)
```

```
{
 int i;
 for(i=0;i<lchrom;i++) lista[talla].cromosoma[i]=recorrido[i];
 lista[talla].objetivo=dato;
 lista[talla].edad=0.0;
 talla++;
}

/*-------------------------------------------------------------*/
/*        ENCUENTRA UN DATO Y LO BORRA                         */
/*-------------------------------------------------------------*/
int borrar(double dato)
{
  int i,tam=0;
  tam=talla;
  for(i=0;i<tam;i++){
   if((lista[i].edad>dato)&&(talla>10)){
     recorre(i);
     talla--;
   }
  }
}

/*-------------------------------------------------------------*/
/*                RECORRE UNA LISTA                            */
/*-------------------------------------------------------------*/
void recorre(int i)
{
 int j;
 for(j=i;j<talla;j++)lista[j]=lista[j+1];
}

/*-------------------------------------------------------------*/
/*          ENCUENTRA UN DATO                                  */
/*-------------------------------------------------------------*/
int busca(double dato)
{
  int i;
  for(i=0;i<talla;i++)
    if(lista[i].objetivo==dato) return 0;

  return 1;
}

/*-------------------------------------------------------------------------------------------*/
/*  ESTA FUNCION COLOCA LA DESENDENCIA,                                                      */
/*  QUE SE OBTUVO MEDIANTE EL CRUZAMIENTO, EN LA NUEVA POBLACION                             */
/*-------------------------------------------------------------------------------------------*/
void coloca()
{
  int i;
  double costo;
    for(i=0;i<lchrom;i++)
      recorrido[i]=intarray1[i];
```

```
   costo=objetivo();
   if(busca(costo)) insertar(costo);

    for(i=0;i<lchrom;i++)
     recorrido[i]=intarray2[i];
    costo=objetivo();
    if(busca(costo)) insertar(costo);

}

/*-------------------------------------------------------------------------*/
/*            GENERA UNA SALIDA EN PANTALLA                                */
/*-------------------------------------------------------------------------*/
report()
{
   if(printstrings==1)
   if (gen==sal_gen*mul){mul++;
     sal_mejor_gen();
    /* writepop(outfp);*/ /* Esta función despliega la población */
                  /* completa */
     }
}

/*-------------------------------------------------------------------------*/
/*            GENERA UNA SALIDA EN PANTALLA                                */
/*-------------------------------------------------------------------------*/
writepop()
{
   int   k,i,j=0;

   for(i=0;i<talla;i++){
     fprintf(fptext,"%3d)  ",j);
     for(k = 0; k < lchrom; k++) fprintf(fptext,"%d ",lista[i].cromosoma[k]);
     fprintf(fptext," %8f| ", lista[i].objetivo);
     fprintf(fptext," %8f| ", lista[i].edad);
     fprintf(fptext," %8d| ", talla);
     fprintf(fptext,"\n");
      j++;
   }
}

/*-------------------------------------------------------------------------------------*/
/* GENERA UNA SALIDA EN PANTALLA , SIN LA RUTA, Y CON LA MEJOR                         */
/* APTITUD                                                                             */
/*-------------------------------------------------------------------------------------*
void sal_mejor_gen()
{
   int   k,i;
     fprintf(fptext,"%3d :  ",gen);
     fprintf(fptext," %8f:  ",bestfit[1].fitness);
     fprintf(fptext," %8f:  ",bestfit[0].fitness);
     fprintf(fptext," %8f:  ",avg);
     fprintf(fptext," %8d:  ", talla);
     fprintf(fptext," n");
```

```
}

/*-------------------------------------------------------------------------*/
/*    GENERA UNA SALIDA DE LOS PARAMETROS DEL AG                           */
/*-------------------------------------------------------------------------*/
void parametros()
{
 int j,
 fprintf(fptext," longitud del cromosoma              %8d \n ",lchrom);
 fprintf(fptext," Máxima edad permitida               %8d|\n ",max  edad);
 fprintf(fptext," Número de cruzamientos              %8d \n ",ncross);
 fprintf(fptext," Número de mutaciones                %8d|\n ",nmutation);
 fprintf(fptext," Número de Generaciones              %8d|\n ",maxgen);
 fprintf(fptext," Probabilidad de cruce              %8f|\n ",pcross);
 fprintf(fptext," Probabilidad de mutacion            %8f|\n ",pmutation);
 fprintf(fptext," Tamaño del T. para elegir supervivientes %d|\n ",tourneysize);
 fprintf(fptext," Numero de supervivientes            %d|\n ",nvivos);
 fprintf(fptext," Mejor Aptitud Global               %f|\n ", aptglobal[1].fitness);
 fprintf(fptext," Encontrada en la generación         %d|\n ",aptglobal[1].generation);
 for (j = 0; j < lchrom; j++)
  fprintf(fptext,"%d ",aptglobal[1].chrom[j]);
 fprintf(fptext,"\n");
 fprintf(fptext," Peor Aptitud Global                %f|\n ", aptglobal[0].fitness);
 fprintf(fptext," Encontrada en la generación         %d \n ",aptglobal[0].generation);
 for (j = 0; j < lchrom; j++)
  fprintf(fptext,"%d ",aptglobal[0].chrom[j]);
}

/*-------------------------------------------------------*/
/*            OX.C                                       */
/*-------------------------------------------------------*/
crossover(int actual)
{
   int    i, J, k, cnt, realcntJcross[2],jcross[2];
   int    start, end, tmpcnt, subcnt;
 for (i=0;i<lchrom;i++){
   intarray1[i]  =  lista[actual].cromosoma[i];
   intraybak1[i]=  lista[actual].cromosoma[i];
   intarray2[i]  =  lista[actual+1].cromosoma[i];
   intarraybak2[i]-  lista[actual+1].cromosoma[i];
   }
   if (flip(pcross)) {
          ncross++;
          J = rnd(0, lchrom - 1);
          k - rnd(0, lchrom - 1);
          start = (J < k) ? J : k;
          end = (J > k) ? J : k;

      jcross[0] = start;
          jcross[1] = end;
          subcnt = 0;
          tmpcnt - 0;
          for (i = start; i <= end; i++)
             randomlst[i] = 1;
```

```
for (i – start; i <– end; i++)
   intarray1[tmpcnt++] = intarraybak1[i];
for (i = end + 1; i < lchrom; i++) {
   for (J = start; J <= end; J++)
          if (intarraybak1[i] == intarraybak2[J]) {
             randomlst[J] = 0;
             break;
          }
   if (J > end)
          tmparray[subcnt++] = intarraybak1[i];
}
for (i – 0; i < start; i++) {
   for (J = start; J <= end; J++)
          if (intarraybak1[i] == intarraybak2[J]) {
             randomlst[J] = 0;
             break;
          }
   if (J > end)
          tmparray[subcnt++] = intarraybak1[i];
}
for (i = start; i <= end; i++)
   if (randomlst[i] –- 0)
          intarray1[tmpcnt++] = intarraybak2[i];
for (i = 0; i < subcnt; i++)
   intarray1[tmpcnt++]    tmparray[i];

/* produce el individuo */
subcnt = 0;
tmpcnt = 0;
for (i = start; i <= end; i++)
   randomlst[i] = 1;
for (i = start; i <= end; i++)
   intarray2[tmpcnt++] = intarraybak2[i];
for (i = end + 1; i < lchrom; i++) {
   for (J = start; J <= end; J++)
          if (intarraybak2[i] == intarraybak1[J]) {
             randomlst[J]    0;
             break;
          }
   if (J > end)
          tmparray[subcnt++] = intarraybak2[i];
}

for (i = 0; i < start; i++) {
   for (J = start; J <= end; J++)
          if (intarraybak2[i] == intarraybak1[J]) {
             randomlst[J] = 0;
             break;
          }
   if (J > end)
          tmparray[subcnt++] = intarraybak2[i];
}
```

```
        for (i = start; i <= end; i++)
          if (randomlst[i] == 0)
                intarray2[tmpcnt++] = intarraybak1[i];
        for (i = 0; i < subcnt; i++)
          intarray2[tmpcnt++] = tmparray[i];
       /*Aqui se asignan a la nueva poblacion*/
       coloca();
  } /*estos individuos no se cruzaron */
  else{
        jcross[0] = 0;
        jcross[1] = 0;
   }
}

/*-------------------------------------------------------------------------------*/
/* ABRE UN ARCHIVO EN MODO TEXTO, APUNTADOR DE ARCHIVO: fp_texto  */
/*-------------------------------------------------------------------------------*/
void nombre_archivo()
{
 char archivo[13];
 fprintf(stdout, " Nombre del Archivo de salida: ");
 fscanf(stdin, "%13s", archivo);
 if ((fptext = fopen(archivo,"wt")) == NULL) {
  fprintf(stderr, " Error al crear el Archivo de salida. \n");
 exit(-1);
 }
}

void pantalla()
{
 int i;
 for(i = 0; i < lchrom; i++) printf("%d ",aptglobal[1].chrom[i]);
  printf("\n Mínimo absoluto----------> %f",minabsoluto);
  printf("\n Máximo tamaño de la poblacion---> %d",talla);
  printf("\n Probabilidad de cruzamiento ------> %f ",pcross);
  printf("\n Probabilidad de mutación  --------> %f ",pmutation);
  printf("\n Máxima edad permitida ----------> %d ",max_edad);
  printf("\n \n");
  getche();
}

/*-------------------------------------------------------------------*/
/*   LISTA DEL TORNEO, POSICION EN LA LISTA          */
/*-------------------------------------------------------------------*/
void preselect()
{
   reset();
   tourneypos = 0;
}

/*-----------------------------------------------*/
/*          SELECCION POR TORNEO          */
/*-----------------------------------------------*/
int selection()
```

```
{
  int    pick, winner, i;
  /* Si los miembros restantes no son los suficientes para un torneo, */
  /* entonces resetear la lista*/
  if ((popsize - tourneypos) < tourneysize) {
        reset();
        tourneypos = 0;
  }
  /* Selecciona las estructuras del tamaño del torneo al azar, y dirige */
  /* un torneo */
  winner = tourneylist[tourneypos];
  for (i = 1; i < tourneysize; i++) {
        pick = tourneylist[i + tourneypos];
        if (lista[pick].objetivo < lista[winner].objetivo)
           winner = pick;
  }
  /* Actualiza la posicion-torneo*/
  tourneypos += tourneysize;
  return (winner);
}

/*--------------------------------------------------------------------------*/
/*  REVUELVE LA LISTA DEL TORNEO AL AZAR                                    */
/*--------------------------------------------------------------------------*/
static void reset()
{
  int    i, rand1, rand2, temp;
  for (i = 0; i < popsize; i++)
        tourneylist[i] = i;
  for (i = 0; i < popsize; i++) {
        rand1 = rnd(i, popsize - 1);
        rand2 = rnd(i, popsize - 1);
        temp = tourneylist[rand1];
        tourneylist[rand1] = tourneylist[rand2];
        tourneylist[rand2] = temp;
  }
}

/*-----------------------------------------------------*/
/*  SELECCIONA LOS SUPERVIVIENTES    */
/*-----------------------------------------------------*/
void supervivientes(void)
{
  int i;
  preselect();
  for(i=0;i<nvivos;i++)
    temp[i]=lista[selection()];
  for(i=0;i<nvivos;i++)
    lista[i]=temp[i];
  talla=nvivos;
  popsize=talla;
}
```

```
/*-------------------------------------------------------------------*/
/*  PRODUCE UN INDIVIDUO ALEATORIAMENTE          */
/*-------------------------------------------------------------------*/
 void revuelve()
 {
   int i,j,k,temp;
   double costo=0.0;
    for(i=0;i<lchrom;i++) recorrido[i]=i;
    for(i=0;i<70;i++){
     j=rnd(0,lchrom-1);
     k=rnd(0,lchrom-1);
     temp=recorrido[j];
     recorrido[j]=recorrido[k];
     recorrido[k]=temp;
    }
    costo=objetivo();
    insertar(costo);
}

/*-----------------------------------------------------*/
/*  GENERA LA POBLACION AL AZAR       */
/*-----------------------------------------------------*/
void pop_azar()
{
  int i;
   for(i=0;i<popsize;i++) revuelve();
}
```

RESUMEN AUTOBIOGRAFICO

Marte Alejandro Castro Fabela

Candidato para el Grado de

Maestro en Ciencias de la Administración con Especialidad en Sistemas

Tesis: ALGORITMOS GENÉTICOS CON EDADES PARA RESOLVER EL PROBLEMA DEL AGENTE VIAJERO

Campo de Estudio: Sistemas

Biografía:

Nacido en Monclova, Coahuila, el 28 de Mayo de 1974, hijo de Felipe de Jesús Castro Sánchez y María Aurelia Fabela de Castro.

Educación:

Egresado de la Facultad de Ingeniería Mecánica y Eléctrica de la Universidad Autónoma de Coahuila; grado obtenido de Ingeniero en Sistemas Computacionales en 1996.

Experiencia Profesional:

Analista de sistemas del departamento de Desarrollo de Sucursales de Banorte y Maestro Instructor de la Coordinación de Educación Continua de la Facultad de Ingeniería Mecánica y Eléctrica de la UANL.