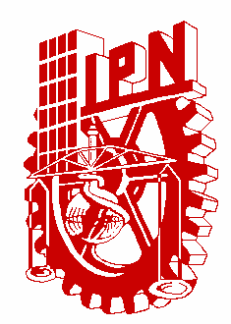


I N S T I T U T O P O L I T E C N I C O N A C I O N A L

Escuela Superior de Ingeniería
Mecánica y Eléctrica: Unidad Zacatenco

Sección de Estudios de Posgrado e Investigación
Programa de Estudios de Posgrado en Ingeniería de Sistemas

“Manipulación Genética aplicada al problema del Agente Viajero”

T E S I S

Que para Obtener el Grado de
Maestro en Ciencias en Ingeniería de Sistemas

P r e s e n t a :

Ing. Victor Manuel Bonifacio Rodríguez González

Director de Tesis:

M. en C. Efraín Martínez Ortiz

México, D.F. Diciembre del 2003

INDICE

INDICE

INSTITUTO POLITECNICO NACIONAL
COORDINACION GENERAL DE POSGRADO E INVESTIGACION

CARTA CESION DE DERECHOS

En la Ciudad de México, Distrito Federal, el día 12 del mes Diciembre del año 2003, el (la) que suscribe Ing. Victor Manuel Bonifacio Rodríguez González alumno(a) del Programa de Maestría en Ciencias con Especialidad en Ingeniería de Sistemas con número de registro 000251, adscrito a la Sección de Estudios de Posgrado e Investigación de la ESIME Unidad Zacatenco, manifiesta que es autor(a) intelectual del presente Trabajo de Tesis bajo la dirección del M. en C. Efraín Martínez Ortiz y cede los derechos del trabajo intitulado: Manipulación Genética Aplicada al Problema del Agente Viajero, al Instituto Politécnico Nacional para su difusión, con fines académicos y de investigación.

Los usuarios de la información no deben reproducir el contenido textual, graficas o datos del trabajo sin el permiso expreso del autor y/o director del trabajo. Este puede ser obtenido escribiendo a la siguiente dirección: bonifax@hotmail.com.
Si el permiso se otorga, el usuario deberá dar el agradecimiento correspondiente y citar la fuente del mismo.

Ing. Victor M. B. Rodríguez Glez.
Nombre y firma

CGPI-14

INSTITUTO POLITECNICO NACIONAL

COORDINACION GENERAL DE POSGRADO E INVESTIGACION

ACTA DE REVISION DE TESIS

En la Ciudad de México, D. F. siendo las 13:00 horas del día 11 del mes de Diciembre del 2003 se reunieron los miembros de la Comisión Revisora de Tesis designada por el Colegio de Profesores de Estudios de Posgrado e Investigación de la E. S. I. M. E. para examinar la tesis de grado titulada:

"MANIPULACION GENETICA APLICADA AL PROBLEMA DEL AGENTE VIAJERO"

Presentada por el alumno:

RODRIGUEZ	**GONZALEZ**	**VICTOR MANUEL BONIFACIO**
Apellido paterno	materno	nombre(s)

Con registro: | 0 | 0 | 0 | 2 | 5 | 1 |

Aspirante al grado de:

MAESTRO EN CIENCIAS

Después de intercambiar opiniones los miembros de la Comisión manifestaron ***SU APROBACION DE LA TESIS***, en virtud de que satisface los requisitos señalados por las disposiciones reglamentarias vigentes.

LA COMISION REVISORA

Director de tesis

M. EN C. EFRAIN MARTINEZ ORTIZ

DR. LUIS MANUEL HERNANDEZ SIMON

M.EN C. ERNESTO MERCADO RAMIREZ

M. EN C. LEOPOLDO GALINDO SORIA

DR. JUAN DE LA CRUZ MEJIA TELLEZ

M. EN C. IGNACIO PEON ESCALANTE

EL PRESIDENTE DEL COLEGIO

DR. FLORENCIO SANCHEZ SILVA

SECCION DE ESTUDIOS DE POSGRADO E INVESTIGACION

Glosario de términos.

Algoritmo Genético.- Conjunto de pasos que imitan el comportamiento genético de los organismos con un mismo objetivo.

AG.- Abreviación de Algoritmo Genético.

AGs.- Abreviación de Algoritmos Genéticos.

Código Genético.- Diccionario empleado para interpretar la información de un organismo a nivel molecular.

Converge.- Concurrir al mismo fin.

Cromosoma.- Se le llama así al conjunto de genes organizados en forma de cadena o ristra. Se encuentra compuesto por varios fenotipos.

Fenotipo.- Conjunto de genes estructurados en forma de cadena o ristra que cumplen una labor específica.

Gen o Gene.- Elemento interno de un cromosoma que por si solo o en conjunto realiza una función biológica específica.

Genoma.- Mapa genético de un organismo.

Genotipo.- Interpretación genética del fenotipo. Es el encargado de darle sentido y valor al conjunto de genes que se encuentran en el fenotipo.

Mapeo.- Encontrar una relación o proyección de algo en otra parte.

MG.- Manipulación genética.

Order Crossover (OX).- Operador genético de cruce en orden.

Partial Mapped Crossover (PMX).- Operador genético de cruce de mapeo parcial.

Retrovirus.- Virus infectado por un gen de una bacteria.

Ristra.- Conjunto de elementos atados uno a continuación de otro.

TSP.- Iniciales en inglés del Problema del Agente Viajero (Travel Salesman Problem).

RESUMEN

Los Algoritmos Genéticos son métodos de búsqueda que imitan el comportamiento evolutivo de los seres vivos (Selección Natural), son usados desde hace cuatro décadas con bastante éxito en diversas disciplinas.

En este trabajo de tesis de Maestría, se propone una nueva técnica de búsqueda como otra alternativa para la solución de problemas de optimización. Esta técnica trabaja en combinación con la técnica empleada por los Algoritmos Genéticos y está basada en el principio de la manipulación genética empleada para resolver problemas en el genoma humano de los seres vivos. Con ella se busca mejorar el Algoritmo Genético tanto en rapidez como en eficacia.

Para comprobar esto, resolveremos el modelo del Agente Viajero (Travel Salesman Problem) aplicado a la trayectoria correspondiente a diez ciudades de la República Mexicana. El problema propuesto se resolverá mediante dos técnicas: la empleada por un Algoritmo Genético y la propuesta en este trabajo. Ambas técnicas serán programadas en una herramienta desarrollada en el lenguaje computacional “C”. Al final ambos resultados serán comparados y comentados y se mostrará las ventajas de la técnica propuesta en el proyecto de tesis.

ABSTRACT

The Genetic Algorithms are methods of search that imitate the evolutive behavior of living organisms (Natural Selection). Since four decades ago they have being used with a certain degree of success in many disciplines.

In this investigation project a new technique of search is proposed like one more for the solution of optimization problems. This technique works in combination with other used with Genetic Algorithms, it is based on genetic manipulation used to solve problems in the human genome. This technique improves the speed and efficiency of the Genetic Algorithm. To verify this, the model of the Travel Salesman Problem in ten cities of Mexico is analized.

The problem will be solved with two techniques, the used for a Genetic Algorithm and the proposed in this work. Both techniques will be programmed in the computational language C. At the end, both results will be contrasted and commented.

INTRODUCCIÓN

En 1859, Darwin publica su libro *"El origen de las especies"*, este libro causó una gran polémica en el mundo científico por las revolucionarias teorías en él contenidas: que las especies evolucionan acorde al medio, para adaptarse a este. Con ello, el universo pasaba de ser una creación de Dios, estática y perfecta desde su inicio, a un conjunto de individuos en constante competencia y evolución para poder perpetuar su especie en el tiempo. La existencia de una especie pasa así a ser algo dinámico; las especies se crean, evolucionan y desaparecen, si no se adaptan. Para cada especie animal, la naturaleza proponía un crudo filtro: sólo los mejores, los más aptos, los que mejor se adapten al medio conseguían sobrevivir lo suficiente para llegar a la madurez, y encontrar una pareja para perpetuar sus aptitudes [16].

La informática ve aquí un claro proceso de optimización. Tomamos los individuos mejor adaptados -mejores soluciones temporales-, los cruzamos -mezclamos-, generando nuevos individuos -nuevas soluciones- que contendrán parte del código genético -información- de sus dos antecesores, y, por lo tanto, aunque el nuevo individuo no tenga que estar forzosamente mejor adaptado -de hecho, puede ser que ni la probabilidad de que el nuevo individuo generado esté mejor adaptado que los padres, sea alta-, el promedio de la adaptación de toda la población; si mejora, ya que tienden a perpetuarse y extenderse las mejores características, y a extinguirse las poco beneficiosas o perjudiciales.

En los últimos años, la comunidad científica internacional ha mostrado un creciente interés en una nueva técnica de búsqueda basada en la teoría de la evolución, que se conoce como el Algoritmo Genético (AG). El algoritmo genético es una técnica de búsqueda basada en la teoría de la evolución de Darwin, que ha cobrado tremenda popularidad en todo el mundo durante los últimos años.

En los Algoritmos Genéticos (AGs), se crea una nueva abstracción – la población- cuya capacidad de adaptación mejorará globalmente, por lo que puede ser que nos encontremos algún individuo con mejores características en generaciones futuras.

El poder de los AGs, proviene del hecho de que se trata de una técnica que se basa en el principio de la selección natural, y pueden tratar con éxito una gran variedad de problemas provenientes de diferentes áreas, incluyendo aquellos en los que otros métodos encuentran dificultades. Si bien no se garantiza que el AG encuentre la solución óptima, del problema, existe evidencia empírica de que se encuentran soluciones de un nivel aceptable, en un tiempo competitivo con el resto de algoritmos de optimización combinatoria.
En el caso de que existan técnicas especializadas para resolver un determinado problema, lo más probable es que superen al AG, tanto en rapidez como en eficacia.

El gran campo de aplicación de los AGs se relaciona con aquellos problemas para los cuales no existen técnicas especializadas. Incluso en el caso en que dichas técnicas existan, y funcionen bien, pueden efectuarse mejoras de las mismas hibridándolas con los AGs.

Los seres vivos o la evolución han resuelto y resuelven innumerables situaciones difíciles. La forma como una especie se comporta frente al medio exterior es la manera que tiene la evolución de resolver un problema.

Supuestamente, los seres vivos evolucionan mediante la selección de los individuos más aptos, apareamiento en el que se produce un cruce de información genética y mutaciones de la información genética. Un modelo informático que emula este proceso mediante operadores genéticos (cruce, mutación, inversión, etc.) puede ser creado con los elementos computacionales adecuados, y es conocido como AG.

Los AGs, se utilizan con bastante éxito en la resolución de problemas de optimización en los cuales el espacio de búsqueda es enorme. Uno de estos problemas es *"El problema del viajero"* (TSP, por sus siglas en inglés, "Travel Salesman Problem"), que es uno de los mas ampliamente estudiados dentro de los problemas de optimización combinatoria [11].

Este problema es conceptualmente muy simple, el viajero debe visitar todas las ciudades de su territorio sólo una vez y volver a la ciudad de partida, las distancias entre cada una de las ciudades es conocida. El objetivo es conseguir que el recorrido del viajero sea el mínimo posible.

El enunciado del TSP, contrasta con la enorme dificultad de su resolución en la práctica al crecer el número de ciudades. Este problema, en su expresión general, es tenido como arquetipo de la complejidad, y se considera que el tiempo necesario para hallar una solución con garantía de óptimo absoluto crece exponencialmente con el número de elementos que intervienen.

Desde mediados de los 50's, un vasto conjunto de literatura sobre el TSP ha sido publicado. El libro escrito por Lawer [15], provee de un perspicaz y comprensivo estudio de los resultados de todas las grandes investigaciones realizadas hasta esa fecha, en 1954, Dantzig y Fulkerson resolvieron el problema para 49 ciudades [11].

En las pasadas dos décadas enormes progresos se han realizado con lo que respecta a optimizar la solución del TSP. Los acontecimientos históricos han sido la solución de problemas con 48, 120, 318, 532, 666 y hasta 2392 ciudades, la más reciente solución se ha dado para un problema de 15,112 ciudades [11].

Los estudios mas recientes que emplean AGs para resolver el TSP, fueron los de Goldberg, Lingle y Grefenstette [8].

Reinelt ha señalado, en su reciente publicación sobre el TSP [3], que a pesar de esos logros, el TSP aún se encuentra lejos de ser resuelto. Muchos aspectos del problema todavía necesitan ser considerados, de igual forma muchas preguntas están aún lejos de ser contestadas satisfactoriamente. Sin embargo, en primera instancia suena interesante imaginar trabajar problemas de optimización como el TSP con AGs que empleen el principio de la "Manipulación Genética", mismo principio que se está usando hoy en día en los laboratorios para corregir las estructuras genéticas dañadas de los seres vivos en particular la del hombre, y reemplazarlas por nuevas.

Al trabajar con información genética es casi de manera implícita hablar de proyectos como el del "Genoma Humano", pensar en el Genoma Humano es referirnos a la manipulación genética, con un poco de suerte para nuestro beneficio y el de todos. El genoma humano, todo el conjunto de genes alojados en 23 pares de cromosomas, constituye nada menos que una autobiografía de nuestra especie. El genoma ha sido corregido, abreviado, modificado y aumentado a medida que se ha transmitido de generación en generación a lo largo de más de tres mil millones de años [12].

Si bien el AG, imita el comportamiento evolutivo de los seres vivos ya desde un punto de vista genético, combinarlo con una técnica que manipula esos genes con un nivel de mayor precisión podría ser bastante beneficioso. Esto es en esencia el propósito de este trabajo de tesis, aplicar nuevas ideas, conceptos y pensamientos, es decir como una nueva versión a una técnica que se ha vuelto un tanto tradicional.

JUSTIFICACIÓN DEL PROYECTO DE TESIS

El estudio del TSP ha sido centro de atención de muchos investigadores de diferentes campos, matemáticas, investigación de operaciones, física, biología e inteligencia artificial, entre otros.

Los operadores genéticos de cruce (“crossover”) y mutación juegan un muy importante papel dentro de los AGs. La aplicación del operador de mutación proporciona diversidad de características entre los individuos de la población, impidiendo la homogeneidad y estancamiento de la misma. El operador de cruce considerado quizás un tanto más importante que los demás o al menos si el más empleado, es el encargado de transportar la información genética de población a población entre generaciones a través de los individuos, buscando mejorar en promedio la calidad de la población conservando las mejores características de cada individuo. Sin embargo esa búsqueda podría tardar un tiempo considerable, además al heredar información, genética de un individuo a otro, no sabemos en absoluto si esta es beneficiosa o perjudicial para la siguiente población, ya que el bloque de información genética a cruzar es seleccionado aleatoriamente.

Considerando lo anterior y conscientes que la manipulación genética hoy en día está más a pie de puerta intentando mejorar las especies, con individuos mejor dotados (Proyecto Genoma), podemos aplicar este principio para mejorar y acelerar la técnica del AG.

En el presente trabajo, se muestra una aplicación de los AGs utilizados como herramienta para hallar la solución a un problema matemático de búsqueda y optimización proponiendo un nuevo tipo de operador genético. Con este nuevo tipo de operador genético se pretende ampliar las posibilidades de búsqueda tras la solución más óptima a los problemas de optimización. De esta forma puede ser considerado como otra alternativa dentro de los operadores ya existentes.

OBJETIVO GENERAL DEL PROYECTO DE TESIS

- Proponer la Manipulación Genética combinada con un Algoritmo Genético como otra alternativa para la búsqueda de la solución óptima a los problemas de optimización.

OBJETIVOS PARTICULARES

- Resolver el problema del Agente Viajero aplicado a un caso real, empleando un Algoritmo Genético convencional.

- Resolver el problema del Agente Viajero aplicado a un caso real, empleando un Algoritmo Genético modificado con el principio de la Manipulación Genética.

- Desarrollar un programa computacional que aplique la técnica usada por un Algoritmo Genético convencional.

- Desarrollar un programa computacional que aplique la técnica usada por un Algoritmo Genético modificado con el principio de la Manipulación Genética.

- Comparar los resultados obtenidos de ambos problemas resueltos.

- Aplicar el principio de la Manipulación Genética combinada con un Algoritmo Genético al problema del Agente Viajero.

METODOLOGÍA APLICADA PARA EL DESARROLLO DEL PROYECTO DE TESIS:

- Formulación del problema.

  El problema que se resolverá en este trabajo de tesis es el Modelo del Agente Viajero, se le dará solución empleando la técnica utilizada por los Algoritmos Genéticos y una técnica basada en el principio de la Manipulación Genética. Capítulo 2 y 3.

- Construcción del modelo.

  El Modelo del Agente Viajero se resolverá para diez ciudades de la República Mexicana, con el objetivo de encontrar el recorrido con la ruta más corta. Capítulo 4.

- Solución del modelo.

  El Modelo del Agente Viajero será resuelto para diez ciudades con la técnica utilizada por los Algoritmos Genéticos y con la técnica basada en el principio de la Manipulación Genética, ambas técnicas serán programadas en el lenguaje computacional Borland C para agilizar los cálculos. Capítulo 4.

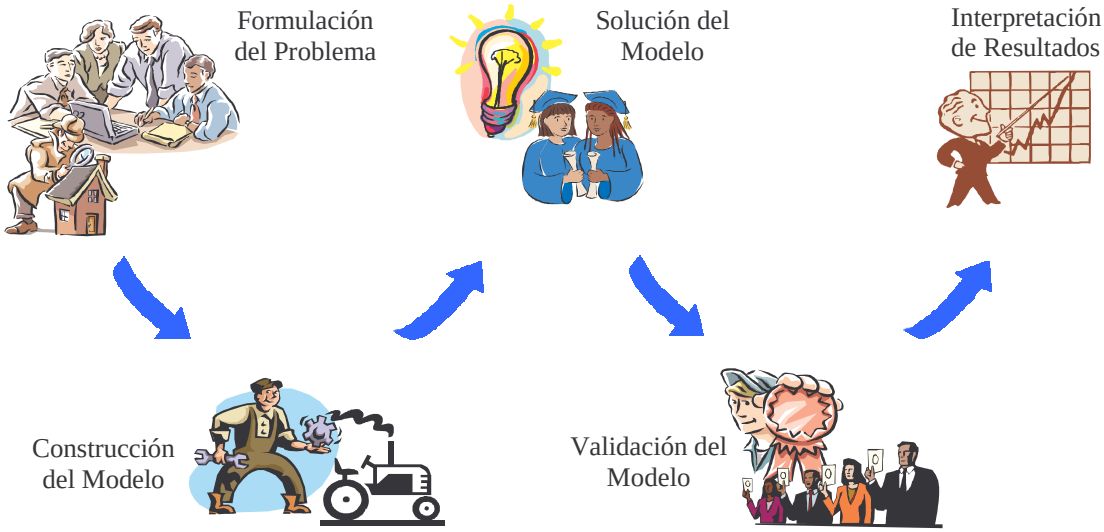
- Validación del modelo.

  El Modelo del Agente Viajero será resuelto también por medio de la búsqueda exhaustiva y por medio de un algoritmo genético convencional, ambos resultados son comparados con los obtenidos con la técnica basada en el principio de la Manipulación Genética. Capítulo 4.

METODOLOGÍA APLICADA PARA EL DESARROLLO DEL PROYECTO DE TESIS:

- Interpretación de los resultados e implicaciones.

  Los resultados son graficados, comentados y discutidos. Se determina la precisión obtenida con la técnica de la manipulación genética. Capítulo 4 y 5.



CAPÍTULO 1.-
Algoritmos Genéticos

1.1 DEFINICIÓN

Los Algoritmos Genéticos (AGs) son métodos adaptativos que pueden usarse para resolver problemas de búsqueda y optimización. Están basados en el proceso genético de los organismos vivos. A lo largo de las generaciones, las poblaciones evolucionan en la naturaleza de acorde con los principios de la selección natural y la supervivencia de los más fuertes, postulados por Darwin. Por imitación de este proceso, los Algoritmos Genéticos son capaces de ir creando soluciones para problemas del mundo real. La evolución de dichas soluciones hacia valores óptimos del problema depende en buena medida de una adecuada codificación de las mismas.

Los principios básicos de los Algoritmos Genéticos fueron establecidos por Holland [3], y se encuentran bien descritos en forma clara en varios textos – Goldberg [7], Davis [4], Michalewicz y Reeves [8].

En la naturaleza, los individuos de una población compiten entre sí en la búsqueda de recursos tales como: comida, agua y refugio. Incluso, los miembros de una misma especie compiten a menudo en la búsqueda de un compañero. Aquellos individuos que tienen más éxito en sobrevivir y en atraer compañeros tienen mayor probabilidad de generar un gran número de descendientes. Por el contrario, individuos poco dotados producirán un menor número de descendientes. Esto significa que los genes de los individuos mejor adaptados se propagarán en sucesivas generaciones hacia un mayor número de individuos. La combinación de buenas características provenientes de diferentes ancestros, puede a veces producir descendientes “superindividuos”, cuya adaptación es mucho mayor que la de cualquiera de sus ancestros.

De esta manera, las especies evolucionan logrando unas características cada vez mejor adaptadas al entorno en el que viven.

Los Algoritmos Genéticos usan una analogía directa con el comportamiento natural. Trabajan con una población de individuos, cada uno de los cuales representa una solución factible a un problema dado. A cada individuo se le asigna un valor o puntuación, relacionado con la bondad de dicha solución.

En la naturaleza, esto equivaldría al grado de efectividad de un organismo para competir por unos determinados recursos. Cuanto mayor sea la adaptación de un individuo al problema, mayor será la probabilidad de que el mismo sea seleccionado para reproducirse, cruzando su material genético con otro individuo seleccionado de igual forma. Este cruce producirá nuevos individuos – descendientes de los anteriores – los cuales comparten algunas de las características de sus padres. Cuanto menor sea la adaptación de un individuo, menor será la probabilidad de que dicho individuo sea seleccionado para la reproducción, y por tanto de que su material genético se propague en sucesivas generaciones.

De esta manera se produce una nueva población de posibles soluciones, la cual reemplaza a la anterior y verifica la interesante propiedad de que contiene una mayor proporción de buenas características en comparación con la población anterior. Así a lo largo de las generaciones las buenas características se propagan a través de la población. Favoreciendo el cruce de los individuos mejor adaptados, van siendo exploradas las áreas más prometedoras del espacio de búsqueda. Si el Algoritmo Genético ha sido bien diseñado, la población convergerá hacia una solución óptima del problema.

En la siguiente figura, se muestra el pseudocódigo de un AG simple, se necesita una codificación o representación del problema, que resulte adecuada al mismo, se requiere una función de ajuste o adaptación al problema, la cual asigna un número a cada posible solución codificada:

```
BEGIN /*Algoritmo Genético*/
        Generar una población inicial
        Computar la función de evaluación de cada individuo
        WHILE NOT Terminado DO
        BEGIN /*Producir nueva generación*/
            FOR Tamaño de la población / DO
            BEGIN /*Ciclo reproductivo*/
                Seleccionar dos individuos de la anterior generación,
                Para el cruce (probabilidad de selección proporcional
                a la función de evaluación del individuo).
                Cruzar con cierta probabilidad los dos individuos
                obteniendo dos descendientes.
                Mutar los dos descendientes con cierta probabilidad.
                Computar la función de evaluación de los dos descendientes
                mutados.
                Insertar los dos descendientes mutados en la nueva generación.
            END
            IF      la población ha convergido THEN
                    Terminado = TRUE
      END
END
```

Figura 1.1 Pseudocódigo de un AG simple.

Durante la ejecución del algoritmo, los padres deben ser seleccionados para la reproducción, a continuación dichos padres seleccionados se cruzarán generando dos hijos, sobre cada uno de los cuales actuará un operador de mutación. El resultado de la combinación de las anteriores funciones será un conjunto de individuos (posibles soluciones al problema), los cuales en la evolución del AG formarán parte de la siguiente población.

1.2 CODIFICACIÓN DE LA POBLACIÓN INICIAL

Se supone que los individuos de la población inicial, que son las posibles soluciones del problema, pueden representarse como un conjunto de parámetros conocidos como *genes*, los cuales agrupados forman una cadena o ristra de valores referida como *cromosomas*. Los genes, son representados por una *serie* de símbolos generalmente pertenecientes a un *alfabeto* por esta razón puede ser empleado cualquier sistema de tipo numérico: decimal, octal, etc.

Si bien el alfabeto empleado para representar los individuos no debe necesariamente estar constituido por el conjunto (0, 1), buena parte de la teoría en la que se fundamentan los AGs utiliza dicho alfabeto.

En la figura 1.2, se muestra la codificación de un individuo basada en ceros y unos.

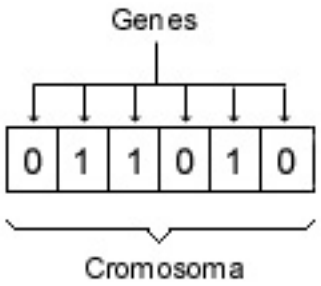


Figura 1.2 Representación de un individuo como cromosoma

Cada elemento de la cadena (gen) le da un sentido diferente al valor en sí del cromosoma de acuerdo a su posición; por eso en la mayoría de los casos, se imita al código Gray, empleado en los circuitos combinatorios. En este caso, se está representando en la solución codificada (cromosoma) el valor de 26 en decimal.

El proceso para generar la población inicial de un AG, se conforma de los siguientes pasos:

1. La población inicial comúnmente es generada de manera aleatoria, constituida por un determinado número de individuos -que por lo regular-

es fijo al paso de generación en generación. Los individuos que forman la población inicial son soluciones factibles, deben ser diferentes entre si.

2. Definición de los criterios de tamaño de la población:

   - Balance entre una población muy pequeña (que implica la convergencia al máximo local) y una población muy grande (que implica muchos recursos de cálculo de operaciones)

   - Aunque, normalmente se elige una población de tamaño fijo, también existen esquemas de poblaciones de tamaño variable.

Figura 1.3

En términos biológicos, el conjunto de parámetros representando un cromosoma particular se denomina *fenotipo*. El fenotipo, contiene la información requerida para construir un organismo, el cual se refiere como *genotipo*; estos mismos términos son empleados en el campo de los AGs. La adaptación al problema de un individuo depende de la evaluación del genotipo; esta última, puede inferirse a partir del fenotipo, es decir puede ser computada a partir del cromosoma, usando la función de evaluación. Dado un cromosoma particular, la función de adaptación le asigna un número real, que se supone refleja el nivel de adaptación al problema del individuo representado por el cromosoma.

1.3 OPERADORES GENÉTICOS

Un algoritmo típico utiliza tres operadores genéticos: *selección*, *cruzamiento* (crossover) y *mutación*, escogidos en gran parte por la analogía con el mundo natural. Estos operadores se encargan de dirigir a la población inicial a través de una serie de pasos, convertidas en generaciones, hacia la convergencia del óptimo global.

1.3.1 OPERADOR GENÉTICO DE: SELECCIÓN

Durante la fase reproductiva se seleccionan los individuos de la población para cruzarse y producir descendientes, que constituirán una vez mutados la siguiente generación de individuos. La selección de padres se efectúa al azar usando un procedimiento que favorezca a los individuos mejor adaptados, ya que a cada individuo se le asigna una probabilidad de ser seleccionado que es proporcional a su función de adaptación.

Este procedimiento se dice que está basado en la “Ruleta Sesgada” [14][Anexo A]. Según dicho esquema, los individuos bien adaptados se escogerán probablemente varias veces por generación, mientras que, los pobremente adaptados al problema no serán seleccionados más que de vez en cuando.

También existen otros procedimientos de selección como el "Torneo" y el "Ranqueo". En el Torneo donde se seleccionan dos individuos aleatoriamente de la población y se opta por el más apto con una probabilidad predeterminada. En el “Ranqueo” se ordena la población por aptitud y se asignan probabilidades de acuerdo a su posición.

Indistintamente, del procedimiento que se emplee para llevar acabo la selección, todos se rigen por la evaluación en la función objetivo la cual determina la aptitud de cada individuo, conociendo así cuáles individuos de la población son los más aptos para ser elegidos. Una vez seleccionados dos padres, sus cromosomas se combinan utilizando habitualmente los operadores de cruce y mutación.

1.3.2 OPERADOR GENÉTICO DE: CRUCE

El operador de cruce tiene una alta probabilidad de ser utilizado y es quizá considerado como el más importante dentro de los AGs, permite la generación de nuevos individuos tomando características de individuos padres.

Existen diferentes tipos de operadores de cruce:

- cruce simple conocido también como operador de cruce de un solo punto.
- cruce de dos puntos.
- cruce uniforme.

El operador de cruce, toma dos padres seleccionados y corta sus ristras de cromosomas en una o varias posiciones escogidas al azar, para producir dos subristras iniciales y dos subristras finales. Después se intercambian las subristras finales, produciéndose dos nuevos cromosomas completos, como se esquematiza en la figura 1.4:

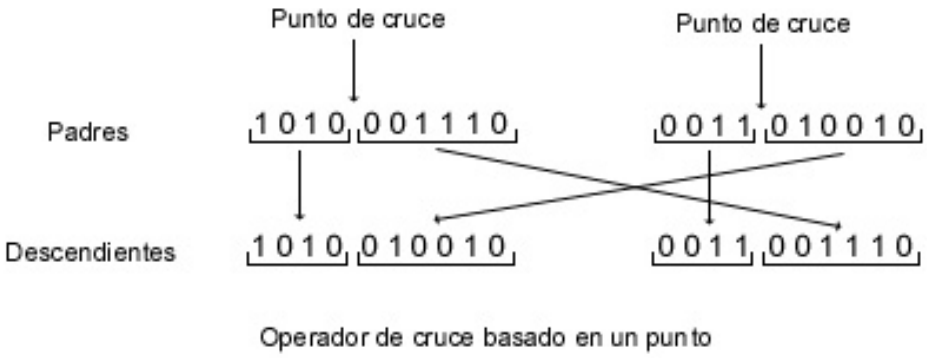


Figura 1.4

Ambos descendientes heredan genes de cada uno de los padres. Por este motivo es sumamente importante que los individuos de la población inicial sean distintos entre sí, es decir que su código genético no sea el mismo.

Habitualmente por estudios e investigaciones de tipo genético, el operador de cruce se aplica de manera aleatoria; normalmente, con una probabilidad comprendida entre 0.5 y 0.1. En el caso en que el operador de cruce no se aplique, la descendencia se obtiene simplemente duplicando los padres.

1.3.3 OPERADOR GENÉTICO DE: MUTACIÓN

El operador genético de mutación puede ser aplicado a todos o algunos miembros de la población (hijo), y consiste en la alteración aleatoria de algún gen componente del cromosoma. La selección del gen es realizada con ayuda de cualquier método que emplee una técnica de selección aleatoria.

En la siguiente figura, se muestra la mutación del cuarto gen del cromosoma.

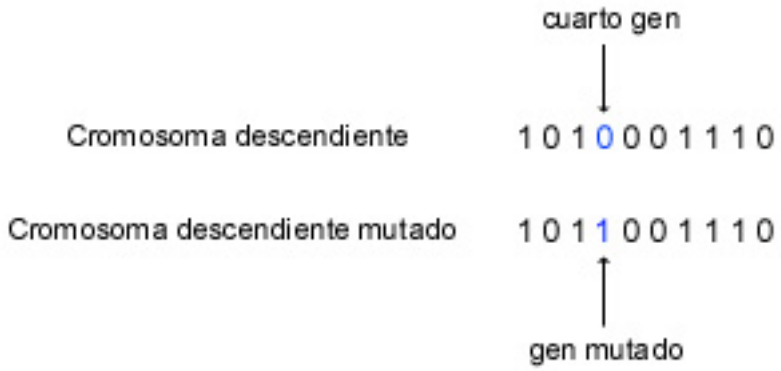


Figura 1.5

Si bien puede -en principio- pensarse que este operador es menos importante que el operador de cruce que, proporciona una exploración rápida del espacio de búsqueda, esto es totalmente erróneo. El operador de mutación asegura que

ningún punto del espacio de búsqueda, tenga probabilidad cero de ser examinado; además, permite introducir nueva información no presente en la población, opera sobre un sólo individuo, determina una posición y la invierte. El operador de mutación es de capital importancia para asegurar la convergencia del algoritmo.

Una vez que se han cruzado y mutado los cromosomas seleccionados, se plantean distintas políticas de sustitución, esto es, qué individuos serán sustituidos en la siguiente generación por la nueva descendencia. Existen dos planteamientos fundamentales a este respecto.

- En el primero los individuos generados sustituyen a sus progenitores, con lo que un individuo no convive nunca con sus antecesores.

- En el segundo planteamiento se sustituyen los individuos peor adaptados de toda la población (por eso se dice que este modelo sigue una política elitista) provocando por tanto que haya convivencia entre un individuo y sus ancestros.

En este momento ya tenemos una nueva generación, generalmente, mejor dotada que la inicial.

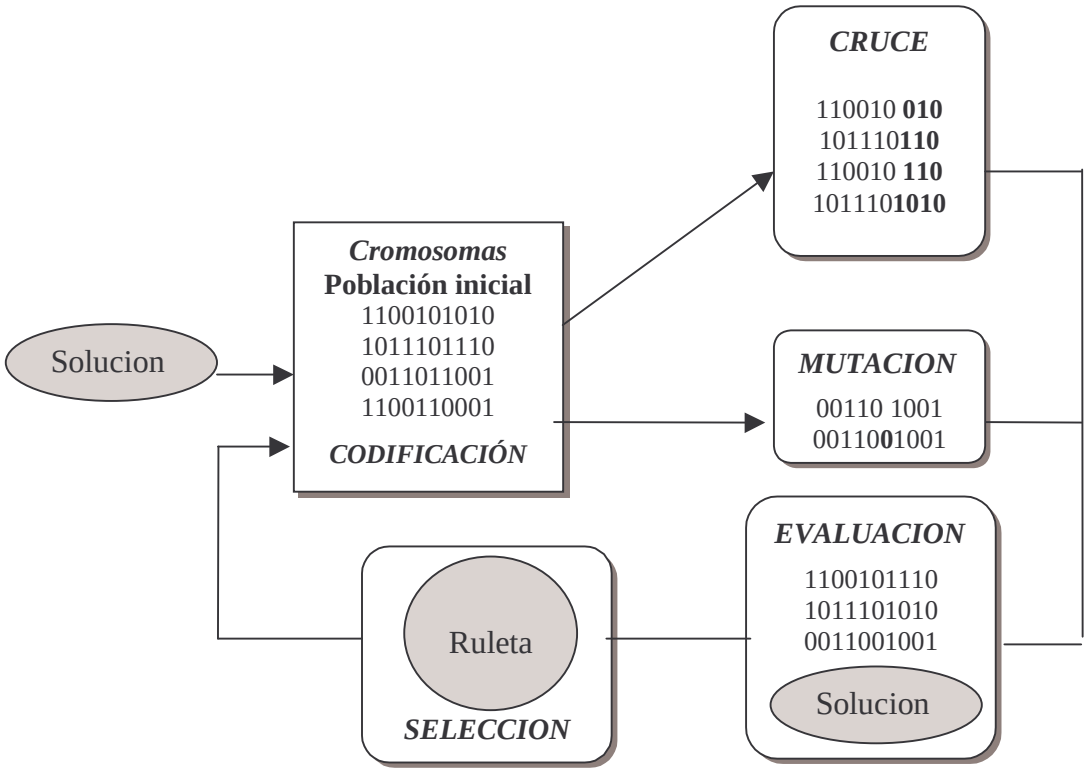


Figura 1.6

Para criterios prácticos, es muy útil la definición de convergencia introducida en este campo por el Dr. De Jong en su tesis doctoral [5]. Si el AG, ha sido correctamente implementado, la población evolucionará a lo largo de las generaciones sucesivas de tal manera que la adaptación media extendida a todos los individuos de la población, así como la adaptación del mejor individuo se irán incrementando hacia el óptimo global.

El concepto de convergencia está relacionado con la progresión hacia la uniformidad: un gen ha convergido cuando al menos el 95% de los individuos de la población comparten el mismo valor para dicho gen. Se dice que la población converge cuando todos los genes han convergido. Se puede generalizar dicha definición al caso en el que al menos algunos de los individuos de la población hayan convergido.

A medida que el número de generaciones aumenta, es más probable que la adaptación media se aproxime a la del mejor individuo.

Si se supiera la respuesta a la que debemos llegar de antemano, entonces detener el algoritmo genético sería algo trivial. Sin embargo, eso casi nunca es posible, por lo tanto se puede decir que se establecen 2 criterios principales de detención:

- ejecutar el algoritmo genético durante un número máximo de generaciones o,

- detenerlo cuando la población se haya estabilizado; es decir, cuando todos o la mayoría de los individuos tengan la misma aptitud.

Hasta el momento en éste capítulo se ha realizado una explicación de lo que son los AG’s, cómo funcionan y los elementos que lo constituyen. El siguiente capítulo está dedicado a un problema de optimización combinatoria, el cual más adelante será objeto de estudio de los AG’s.

CAPÍTULO 2.-
El problema del Agente Viajero (TSP)

2.1 PLANTEAMIENTO DEL PROBLEMA

El Problema del Viajero (TSP) como se mencionó al inicio es sencillo de comprender, el viajero debe realizar un recorrido en el cual debe de visitar todas las ciudades de su territorio solo una vez y regresar a la ciudad considerada como punto de partida. Típicamente, las reglas son que ninguna ciudad es visitada más de una vez.

Entre cada una de las ciudades existe una distancia que las separa de las demás la cual es conocida, la meta es encontrar el recorrido más corto es decir, aquel que involucre la menor distancia.

A cada uno de los recorridos que puede realizar el viajero se le denominará viaje. Comprobar cada uno de los posibles viajes para *n* ciudades tendría un costo de *n* factorial sumas. Si cada viaje tiene 30 ciudades en total serán $2.65x10^{32}$ sumas. Considerando una media de 1 billón de sumas por segundo, esto corresponde a 8,000,000,000,000,000 años. El solo hecho de añadir sólo una ciudad más causaría que el número de sumas se incrementase por un factor de 31. Lógicamente, esto es una solución prácticamente imposible.

El estudio de este problema ha llamado la atención de muchos investigadores de diferentes campos, esto se debe a que el TSP exhibe todos los aspectos de la optimización combinatoria y ha servido, y continúa sirviendo a problemas como el del banquero para nuevos algoritmos de simulación, redes neuronales y métodos evolutivos. Otros ejemplos de problemas de optimización combinatoria son colocación de tuberías de gas y agua, diseño estructurado. Los estudios más recientes que emplean algoritmos genéticos para resolver el TSP fueron los de Goldberg, Lingle y Grefenstette [8]. Desde entonces, el TSP ha sido objeto de estudio para la comunidad que trabaja con los algoritmos genéticos.
Los estudios del TSP a través de los algoritmos genéticos proveen de ricas experiencias y muestran las bases de los problemas de optimización combinatoria. Estrictamente hablando, los mayores esfuerzos que se han realizado han sido para lograr lo siguiente:

1.- Ofrecer una apropiada representación codificada del recorrido.

2.- Conseguir operadores genéticos aplicables para mantener las subcadenas obtenidas y eliminar la ilegalidad de las mismas.

3.- Prevenir la convergencia prematura.

Grandes esfuerzos han sido llevados acabo para tratar de encontrar un algoritmo eficiente para la solución de problemas como estos, lo cual ha llevado a la introducción de los Algoritmos Genéticos como herramienta.

El principal problema en la aplicación de AGs para muchos es que el operador de cruce y el operador de mutación presentan un gran potencial para crear recorridos que no son factibles.

Un AG en comparación con otros métodos puede encontrar una solución en mucho menos tiempo, aunque probablemente no sea la mejor solución, podrá encontrar una buena solución. Hay dos pasos básicos para resolver este problema utilizando algoritmos genéticos:

- El primero es crear un grupo de viajes, aleatoriamente, al que se le llama nuestra población inicial. Estos viajes serán almacenados como una secuencia de números.

- El segundo paso es encontrar los dos mejores en la población y combinarlos para crear dos nuevos viajes hijos que se espera sean mejores.

2.2 CODIFICACIÓN DE LA POBLACIÓN INICIAL

Muchos problemas no requieren de optimizar series de parámetros de valores reales pero el descubrimiento de una lista ideal ordenada da la pauta al clásico ejemplo del TSP.

Tradicionalmente, los cromosomas son una simple cadena binaria, esta simple cadena no es adecuada para el TSP o para otro problema combinatorio. En los últimos años, diversos esquemas de codificación han sido propuestos para el TSP, entre ellos el de “permuta” y “llaves aleatorias”; han tenido una aceptación no solo para el TSP sino también para otros problemas de optimización combinatoria La codificación se realiza en forma de vector, siguiendo el orden del recorrido.

En la siguiente figura, se muestra un ejemplo de codificación. Se tiene una población de cinco ciudades las cuales se encuentran etiquetadas por un número entero positivo para facilitar su ubicación y reconocimiento.

El recorrido que se muestra es el 3-2-5-4-1 y es realizado en un orden aleatorio. En términos genéticos podemos decir que se ha generado un cromosoma de cinco ciudades.

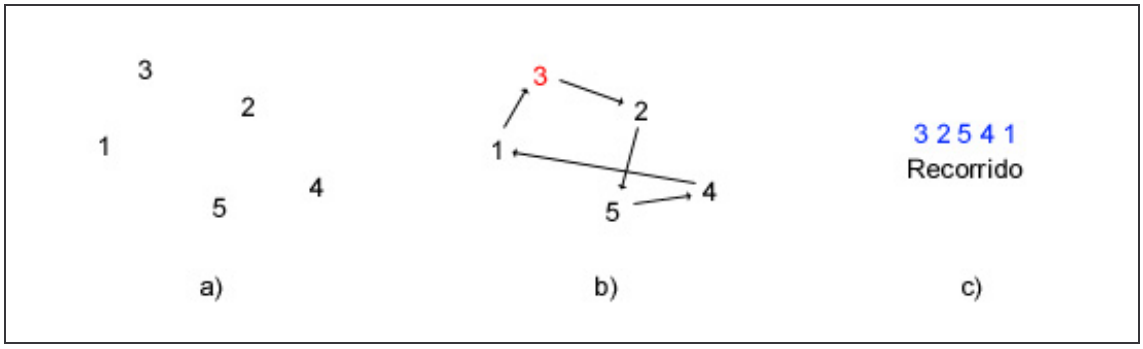


Figura 2.1

No se debe de olvidar que el recorrido finaliza donde se inició es decir, la ciudad que en el recorrido ocupa el lugar de punto de partida también toma el lugar de punto final del mismo. En esta ocasión la ciudad 3, ocupa ese papel.

De la misma forma las ciudades pueden ser etiquetadas con letras de cualquier alfabeto, símbolos, etc.

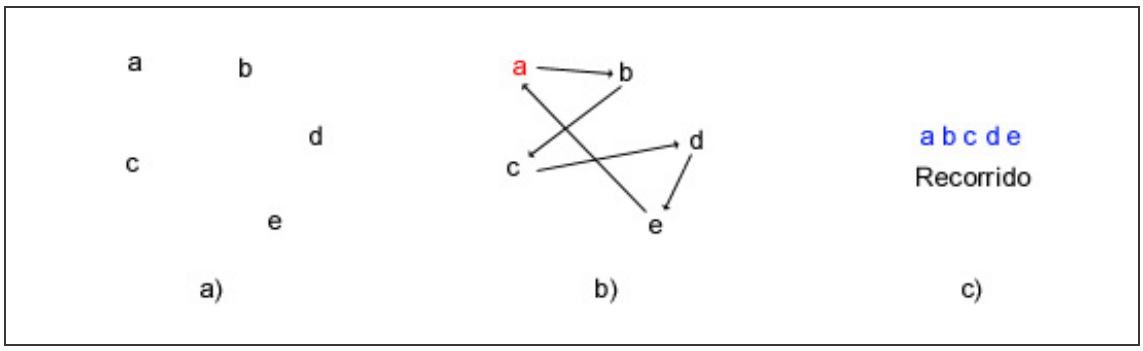


Figura 2.2

En el recorrido anterior la ciudad “a” es tomada como inicio y fin del recorrido.

Otra forma de llevar acabo la codificación, es la de asignar a cada ciudad un valor entre 0 y 1 aleatoriamente.

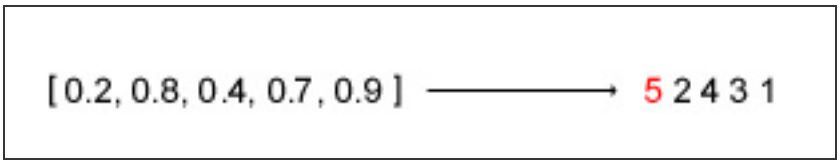


Figura 2.3

El recorrido se obtiene al ordenar estos números de mayor a menor. Los números corresponden a la posición inicial de los valores generados de manera aleatoria.

2.3 APLICACIÓN DE LOS OPERADORES GENÉTICOS AL TSP

2.3.1 OPERADOR GENÉTICO DE CRUCE

En las pasadas décadas, grandes operadores de cruce han sido propuestos tales como:

1.- El cruce de mapeo parcial (PMX).
2.- El cruce en orden (OX).
3.- El cruce en ciclo (CX).
4.- El cruce en posición establecida.
5.- El cruce en orden establecido.
6.- De tipo heurístico, entre otros.

Además, los operadores de cruce pueden ser clasificados en dos clases:

- Aproximación canónica y aproximación heurística

El enorme riesgo que se corre al aplicar al conjunto de soluciones una codificación distinta al código Gray, es la gran probabilidad de crear recorridos no válidos en un futuro. Veamos el siguiente ejemplo:

b c d e g h f a

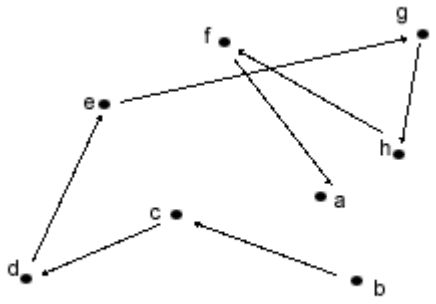


Figura 2.4 Recorrido Padre 1

otro

c b g f a d e h

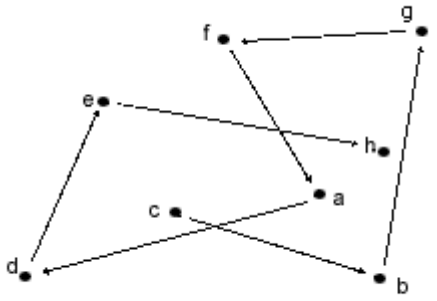


Figura 2.5 Recorrido Padre 2

Si un operador de cruce de punto sencillo es aplicado directamente sobre estos recorridos, por la mitad el resultado sería el siguiente:

b c d e a d e h

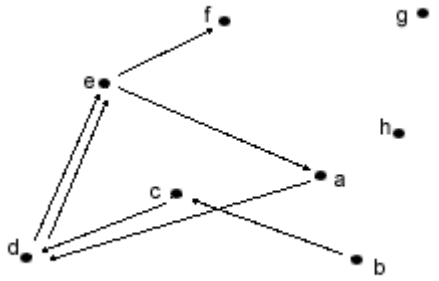


Figura 2.6 Recorrido Hijo 1 Incompleto

y

c b g f g h f a

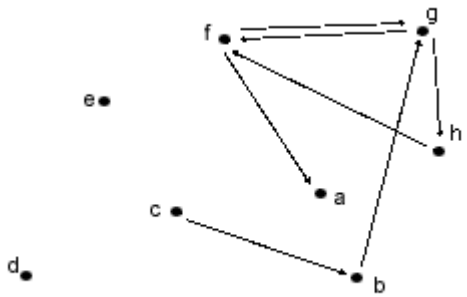


Figura 2.7 Recorrido Hijo 2 Incompleto

Es claro que ninguno de ellos representa un recorrido completo.

Los operadores de aproximación canónica pueden ser vistos como una extensión del operador de cruce de dos puntos o multipuntos para cadenas binarias. Generalmente al emplear este tipo de operadores, algunas ciudades llegan a ser perdidas durante el corte mientras que algunas otras son repetidas. Para evitar esto se ha incluido un proceso de reparación dentro de este tipo de aproximación,

de esta forma se garantiza que ninguna ciudad es excluida del recorrido de la misma manera, ninguna ciudad es visitada mas de una vez, sin olvidar que la longitud de la cadena codificada o cromosoma se mantiene invariable.

La esencia de los operadores de aproximación canónica es el mecanismo aleatorio de búsqueda a ciegas, ya que no hay garantía de que un descendiente producido por este tipo de cruce sea mejor que sus padres. En cambio, las aproximaciones de tipo heurística pretenden generar descendientes improvisados.

A continuación se describe el funcionamiento de los operadores de aproximación canónica.

2.3.2 TÉCNICAS DE CRUCE

- **Cruce de Mapeo Parcial (PMX) por sus siglas en ingles de: Partial-Mapped Crossover)**

El PMX fue propuesto por Goldberg y Lingle [7]. El PMX puede ser visto como una extensión del método crossover de dos puntos aplicado a una cadena binaria. Emplea un método especial de reparación para resolver la ilegitimidad causada por el simple método crossover de dos puntos. Así que lo esencial del PMX es un simple método "crossover" de dos puntos junto con un método de reparación.

Procedimiento de aplicación:

Paso 1. Seleccione dos posiciones a lo largo de la cadena uniformemente al azar. Las subcadenas definidas por las dos posiciones son llamadas las secciones de mapeo.

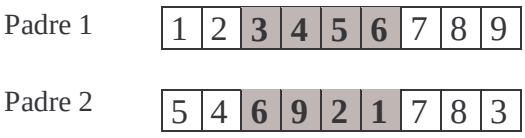
Paso 2. Intercambie dos subcadenas entre padres para producir un proto-hijo.

Paso 3. Determine la relación de mapeo entre las dos secciones de mapeo.

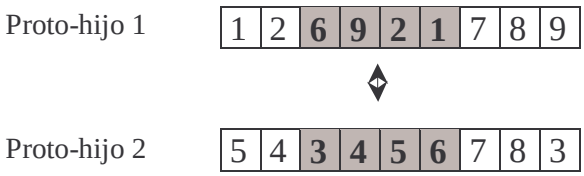
Paso 4. Legalice la descendencia con la relación de mapeo.

Este procedimiento está ilustrado en la siguiente figura. Las ciudades 1, 2 y 9 son duplicadas en el proto-hijo 1, mientras que las ciudades 3, 4, y 5 no se encuentran. De acuerdo a la relación de mapeo establecida en el paso 3, las ciudades que están repetidas 1, 2 y 9 deben ser reemplazadas por las ciudades que faltan 3, 5 y 4 respectivamente.

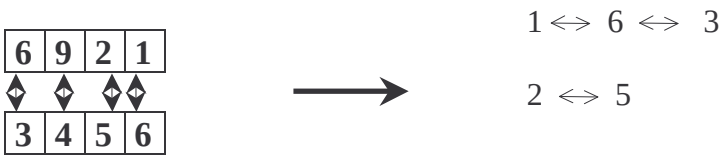
1. Seleccione la subcadena al azar.



2. Cambie las subcadenas entre padres



3. Determine la relación de mapeo



4. Legalice la descendencia con la relación de mapeo

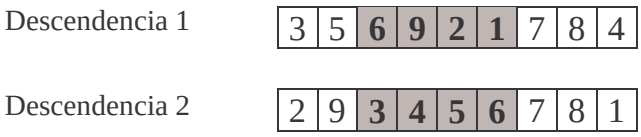


Figura 2.8 Aplicación del operador PMX.

❖ Cruce en Orden (OX) por sus siglas en ingles de: Order Crossover.

OX fue propuesto por Davis [4]. Este puede ser visto como una clase de variación del PMX con un diferente método de reparación. OX trabaja de la siguiente manera:

Procedimiento:

Paso 1. Seleccione una subcadena de un padre al azar

Paso 2. Se produce un proto-hijo mediante la copia de la subcadena dentro de la correspondiente posición de este.

Paso 3. En el segundo padre borre las ciudades que están ya en la subcadena. La secuencia resultada de ciudades contiene las ciudades que el proto-hijo necesita.

Paso 4. Coloque las ciudades dentro de las posiciones sin arreglo del proto-hijo de izquierda a derecha de acuerdo a la orden de la secuencia para producir una descendencia.

El procedimiento se ilustra en la siguiente figura, la cual nos da un ejemplo de cómo hacer una descendencia. Con los mismos pasos, nosotros podemos producir la segunda descendencia como [2 5 4 9 1 3 6 7 8], de los mismos padres:

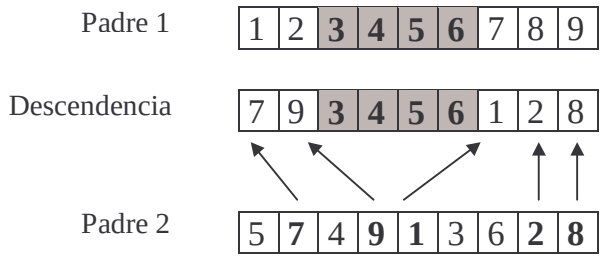


Figura 2.9 Aplicación del operador OX.

❖ Cruce en Posición Establecida

Fue propuesto por Syswerda [13]. Esto es esencialmente una clase de "crossover" uniforme para la representación de una permutación junto con un método de reparación. Esto también puede ser visto como una clase de variación del OX en el cual las ciudades están seleccionadas inconsecutivamente. El operador de cruce "Position-Based Crossover" trabaja de la siguiente manera:

Procedimiento de aplicación:

Paso 1. Seleccione un juego de posiciones de un padre aleatoriamente.

Paso 2. Produzca un proto-hijo mediante la copia de las ciudades sobre estas posiciones dentro de las posiciones correspondientes del proto-hijo.

Paso 3. Borre las ciudades las cuales ya fueron seleccionadas del segundo padre. La secuencia resultada de ciudades contiene las ciudades que el proto-hijo necesita.

Paso 4. Coloque las ciudades dentro de las posiciones sin arreglo del proto-hijo de izquierda a derecha de acuerdo a la orden de la secuencia para producir una descendencia.

Este procedimiento se muestra en la siguiente figura. Con los mismos pasos nosotros podemos producir la segunda descendencia como [2 4 3 5 1 9 6 7 8]

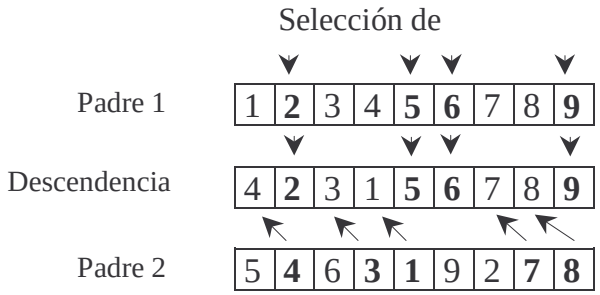


Figura 2.10 Aplicación del operador de Posición Establecida.

❖ Cruce en Orden Establecido

También fue propuesto por Syswerda [13]. Este tiene una ligera variación del "Position-Based Crossover", en el cual el orden de las ciudades en la posición seleccionada en un padre es impuesto sobre las correspondientes ciudades en el otro padre, como se muestra en la siguiente figura:

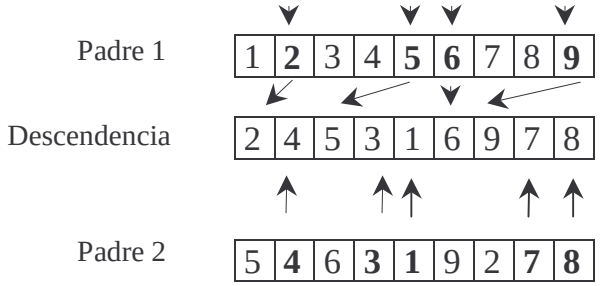


Figura 2.11 Aplicación del operador de Orden Establecido.

Con los mismos pasos nosotros podemos producir la segunda descendencia como [4 2 3 1 5 6 7 9 8].

❖ Cruce en Ciclo (CX) por sus siglas en ingles de: Cycle Crossover.

Fue propuesto por Olive, Smith y Holland [10]. Así como en "Positicon-Based Crossover", CX toma algunas ciudades de un padre y selecciona las restantes ciudades del otro padre. La diferencia es que las ciudades del primer padre no son seleccionadas aleatoriamente sino que en este caso se define un ciclo el cual decide que ciudades con sus posiciones serán seleccionadas. CX trabaja de la siguiente manera:

Procedimiento de aplicación:

Paso 1. Encuentre el ciclo con el cual define la correspondencia de posiciones entre las ciudades de los dos padres

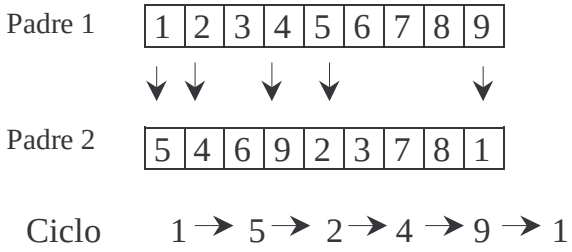
Paso 2. Copie las ciudades en el ciclo para un hijo con la correspondencia de posiciones de un padre.

Paso 3. Determine las ciudades restantes para el hijo mediante la eliminación de estas ciudades las cuales ya están en el ciclo del otro padre.

Paso 4. A completar el hijo con las ciudades restantes.

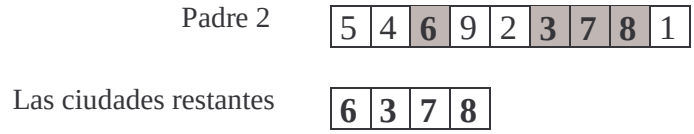
Este procedimiento se ilustra en la siguiente figura. Con los mismos pasos, nosotros podemos producir la segunda descendencia como [5 4 3 9 2 6 7 8 1]:

1. Encuentra el ciclo definido por los padres



2. Copiar al hijo las ciudades en el ciclo

3. Determinar las ciudades restantes del hijo



4. A completar

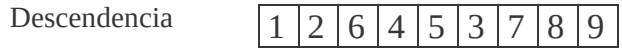


Figura 2.12 Aplicación del operador de Cruce en Ciclo.

❖ Cruce de Intercambio de Subrecorrido

Este operador fue propuesto por Yamamura, Ono y Kobayashi [9]. Se selecciona una subcadena del primer padre, se busca una subcadena en el segundo padre que contenga las mismas ciudades. Las descendencias son creadas mediante intercambios de subcadenas como se muestra en la figura:

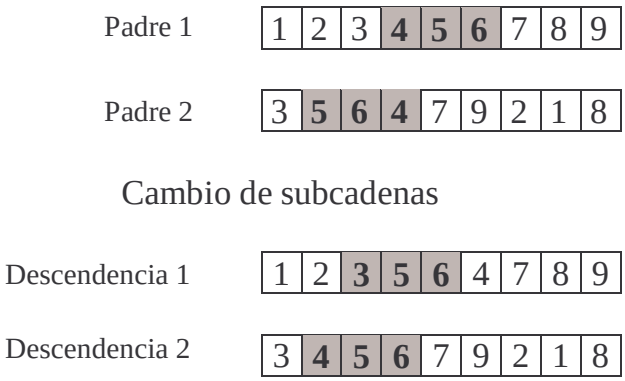


Figura 2.13 Aplicación del operador de Intercambio de Subrecorrido.

Mapeo Parcial (PMX)	**En Orden (OX)**	**En Posición Establecida**
▪ Es una variación del operador de cruce de 2 puntos. ▪ Contiene un procedimiento especial de autoajuste basado en el mapeo, para solucionar legalizar recorridos no válidos.	▪ Es una variación del PMX. ▪ Su procedimiento de autoajuste es diferente al del PMX, se basa en un relleno ordenado del cromosoma.	▪ Es de cruce uniforme y se puede ver como una variación del OX. ▪ A diferencia de OX las ciudades que se cruzan no son seleccionadas en orden.
En Orden Establecido	**En Ciclo (CX)**	**De Intercambio de Subrecorrido**
▪ Es una pequeña variación del cruce en Posición Establecida. ▪ El procedimiento de autoajuste es el mismo que se emplea en el de Posición Establecida.	▪ Al igual que en Posición Establecida, las ciudades a cruzar se toman de primer padre y se seleccionan las restantes del segundo padre. La diferencia es que la selección no es aleatoria es mediante un ciclo que se determina por ambos padres.	▪ Utiliza el operador de cruce de dos puntos. ▪ Selecciona una subcadena en cada padre y se intercambian. La diferencia con PMX es que ambas subcadenas contienen los mismos hijos en común.

Tabla 2.1 Tipo de Operadores de Cruce

Como se puede ver en la tabla y en la descripción de cada operador de cruce en su respectiva sección, hay una gran dependencia entre cada uno de ellos por que mantienen una relación bastante estrecha entre sí, unas técnicas se derivan de otras y se complementan con una tercera. Sus características son prácticamente las mismas, operan haciendo un cruce y poseen un procedimiento de autoajuste para evitar la creación de recorridos con ciudades repetidas o recorridos con ausencia de ciudades. Los más comúnmente empleados son el OX y el PMX, ya que poseen la mayoría de las características que envuelven a los demás operadores.

2.3.3 OPERADOR GENÉTICO DE MUTACIÓN

Relativamente es sencillo generar operadores de mutación, al igual que en los operadores de cruce; en los últimos años han sido propuestos diversos operadores, como son los de: *inversión*, *inserción*, *desplazamiento* e *intercambio recíproco.*

2.3.4 TÉCNICAS DE MUTACIÓN

- **Inversión**

En este método se fijan dos posiciones dentro del cromosoma, se toma la subcadena seleccionada y se invierte volviéndose a insertar en el cromosoma. La selección de las dos posiciones se hace de manera aleatoria.

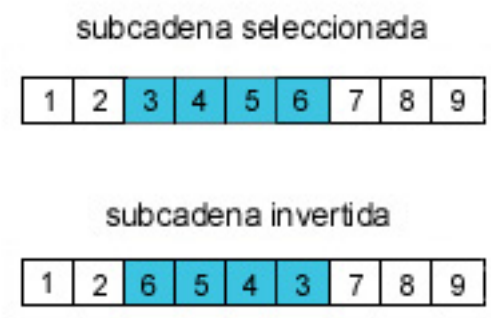


Figura 2.14 Inversión

- **Inserción**

Se selecciona una ciudad del cromosoma de manera aleatoria y se inserta en una posición aleatoria, sin alguna alteración más al orden de los genes del cromosoma.

Figura 2.15 Inserción

- **Desplazamiento**

Una subcadena es seleccionada de manera aleatoria e insertada de la misma manera, dentro del mismo cromosoma. Es el mismo procedimiento que en la inserción, solo que en aquella la subcadena contiene una sóla ciudad.

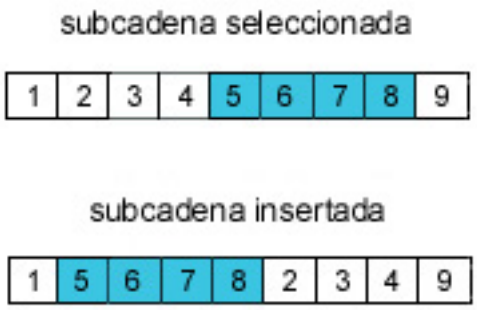


Figura 2.16 Desplazamiento

- **Intercambio Recíproco**

Dos ciudades son seleccionadas de manera aleatoria y son intercambiadas sus posiciones.

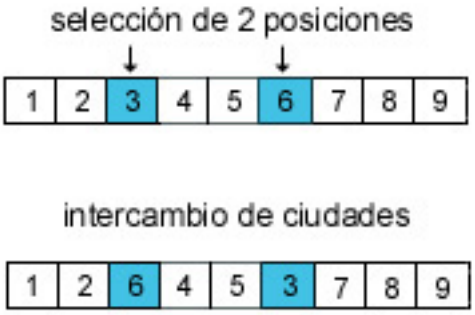


Figura 2.17 Intercambio Recíproco

Dentro de los operadores de mutación se encuentran los de tipo heurístico. La mutación heurística fue propuesta por Cheng y Gen [6], está diseñada con la

técnica del "vecindario" con el fin de producir una descendencia improvisada con características mejores a las de los padres.

Procedimiento para los operadores de tipo heurístico:

Paso 1. Seleccionar la *n* cantidad de genes de manera aleatoria.

Paso 2. Generar vecinos de acuerdo a todas las posibles permutaciones de los genes seleccionados.

Paso 3. Evaluar todos los vecinos y seleccionar uno de los mejores como descendencia

Figura 2.18 Operador de tipo Heurístico.

Como en todo problema resuelto a través de algoritmos genéticos, los progenitores son sustituidos por sus descendientes. Los recorridos resultantes tras la aplicación de los operadores genéticos son evaluados, y aquellos que ofrecen la ruta más óptima y viable son seleccionados para sustituir a los recorridos padres.

Del mismo modo, generación tras generación son evaluados todos los recorridos llegando así a encontrar o aproximarse al mejor posible.

Antes de dar fin a éste capítulo, se debe mencionar que el TSP es un problema típico dentro de la Optimización Matemática. Ha sido seleccionado para ser usado en éste trabajo de tesis por ser un problema bastante ilustrativo y claro en su definición que se acopla muy bien a la mecánica de los AG's.

En el siguiente capítulo se detalla paso a paso la técnica de la Manipulación Genética, su origen y su propósito.

CAPÍTULO 3.-
Técnica Propuesta para resolver el TSP

3.1 ANÁLISIS A LOS OPERADORES GENÉTICOS DE CRUCE DEL PROBLEMA DEL AGENTE VIAJERO

En capítulos anteriores, se ha mencionado que los AGs han sido empleados en diversas disciplinas obteniendo satisfactorios resultados. No hay duda alguna que es un método que funciona correctamente cuando es aplicado de igual forma; pero para esto es sumamente importante utilizar los operadores genéticos de manera adecuada.

Siempre será favorable emplear la codificación adecuada para representar a los miembros de la población, la mutación es un operador que debe emplearse en el momento justo que sea requerido, un mal uso de él podría perjudicar al miembro sobre el cuál es aplicado afectando así a la larga el promedio de calidad de la población. Aplicar un tipo de operador de cruce ("crossover") de manera incorrecta puede provocar para el caso del TSP ilegitimidad en los recorridos.

De los operadores genéticos que conforman el AG, uno de los más importantes es el operador de cruce. Su importancia radica principalmente en ser el encargado de intercambiar información genética entre un miembro de la población y otro, en otras palabras es el responsable de transmitir la herencia de generación en generación. La información genética es transportada en paquetes o subristras, obtenidas del recorrido que describe al individuo o genéticamente hablando, de su cromosoma. La longitud de estos paquetes es variable.

Cuando se intercambian los paquetes conocemos su longitud, su procedencia y su destino; sin embargo, no se conoce la calidad del contenido genético que hay dentro de él, es decir se sabe qué ciudades o genes se están intercambiando pero lo que no se sabe, es si la secuencia de estas ciudades favorezca o perjudique a la siguiente generación. En genética se estaría hablando de heredar genes defectuosos o con alguna deficiencia. Veamos un ejemplo del TSP donde las ciudades son representadas por letras del abecedario.

En la siguiente figura se encuentran dos miembros de una población que han sido seleccionados para ser cruzados y procrear dos nuevos miembros de la siguiente generación.

Individuo 1 B D E A C H G F
Individuo 2 G A B C F H E D

El operador de cruce empleado es el PMX.

B D **E A C H** G F → B D **B C F H** G F → E D **B C F H** G A
G A **B C F H** E D → G A **E A C H** E D → G F **E A C H** B D

Figura 3.1 Cruce entre dos cadenas empleando PMX.

Uno de los paquetes que se está heredando de un individuo de la población a otro es el **BCFH.**

En la siguiente tabla se encuentra la relación de distancias que existe entre cada una de las ciudades involucradas. En ella se puede observar que la relación que hay entre la ciudad **C** y **F** es decir su distancia, sobresale por su alto valor.

	A	**B**	**C**	**D**	**E**	**F**	**G**	**H**
A	0							
B	345	0						
C	543	76	0					
D	**6678**	34	87	0				
E	574	54	562	1547	0			
F	847	823	**7364**	54	345	0		
G	36	2357	354	697	354	1987	0	
H	86	357	87	871	2587	368	785	0

Tabla 3.1 Relación de distancias entre ciudades.

Si se continuara transmitiendo esa relación de generación en generación, es claro que difícilmente nuestra población mejorará, por mucho que las demás relaciones entre el resto de ciudades dentro del mismo recorrido mejoren. La probabilidad de que el recorrido sea el óptimo es baja debido al alto valor de la distancia comprendida entre **C** y **F**.

Otro caso es la relación que aparece en la tabla entre **A** y **D**. El valor de la distancia comprendida entre estas dos ciudades también es demasiado grande, por lo tanto, si en algún momento que se lleva acabo la aplicación del AG se

llegará a presentar esta relación, la convergencia al recorrido óptimo se vería afectada.

Podemos concluir esta sección; generalizando, que siempre que se estén cruzando o intercambiando paquetes de información genética independientemente del operador de cruce empleado, entre dos organismos diferentes, se correrá el riesgo de arrastrar una o más relaciones entre genes que perjudiquen al nuevo organismo creado.

A continuación veamos como la ciencia a través de la Ingeniería Genética, está enfrentando y solucionando estas situaciones.

3.2 MANIPULACIÓN GENÉTICA

En el amanecer del tercer milenio, se está por primera vez en situación de editar el texto de nuestro código genético. Ya no es solo un manuscrito precioso; está en un disco. Se pueden eliminar fragmentos, añadirlos, reorganizar los párrafos o escribir sobre palabras [6].

Para la mayoría de los profanos, el destino es obvio hacia el que se dirige la investigación genética, el premio definitivo por así decirlo, es un ser humano creado mediante Ingeniería Genética. Un día, de aquí a unos siglos, eso podría significar un ser humano con genes recién inventados. De momento, significa un ser humano con un gen que ya existe tomado prestado de otro ser humano, o de un animal o de una planta. ¿Es posible tal cosa? Y si es posible, ¿es ético?

Consideremos un gen del cromosoma que suprime al cáncer de colon (cromosoma 18), se creía que era un gen llamado DCC, pero ahora se sabe que el DCC dirige el desarrollo de los nervios en la columna vertebral y que no tiene nada que ver con la supresión de tumores. El gen supresor de tumores está próximo al DCC, pero sigue siendo escurridizo. Si se nace con una versión defectuosa de este gen. El riesgo de padecer cáncer aumenta considerablemente. ¿Podría eliminarlo un futuro ingeniero genético, como si fuera la bujía defectuosa de un coche, y reemplazarlo? Muy pronto, la respuesta será sí [12].

Un periodista de antaño realizaba su trabajo cortando papel con unas tijeras auténticas y pegándolo con un pegamento auténtico. Hoy en día, para trasladar párrafos de un sitio a otro, se emplean unos pequeños iconos de software decorados convenientemente en un procesador de palabras computacional para indicar que hacen la misma tarea. Pero el principio es el mismo: para trasladar un texto, se corta y se pega en otro sitio.

Hacer lo mismo en el texto de los genes exige también tijeras y pegamento. En ambos casos, afortunadamente, la naturaleza ya los había inventado para sus propios objetivos. El pegamento es una enzima llamada ligasa que une frases

sueltas del ADN cuando se cruza con ellas. Las tijeras, llamadas enzimas de restricción, se descubrieron en las bacterias en 1968 [1].

En 1972, Paul Berg, de la Universidad de Stanford, utilizó enzimas de restricción para partir por la mitad dos trozos de ADN viral en un tubo de ensayo; luego utilizó ligasas para volverlos a reunir formando nuevas combinaciones. Así, se produjo el primer ADN "recombinante" sintético mediante la manipulación genética. La humanidad podría realizar ahora lo que han estado haciendo los retrovirus durante mucho tiempo: insertar un gen en un cromosoma. Al cabo de un año aparecía la primera bacteria creada mediante Ingeniería Genética: una bacteria intestinal infectada con un gen extraído de un anfibio [1].

Inmediatamente, se produjo una oleada de preocupación pública que no se limitó a los profanos. Los propios científicos creyeron conveniente hacer una pausa antes de apresurarse a explorar la nueva tecnología. En 1974, declararon una moratoria sobre todos los experimentos de ingeniería genética que sólo avivó las llamadas de la inquietud pública: si los científicos estaban lo bastante preocupados como para detenerse, entonces verdaderamente debía haber algo por lo que preocuparse. La naturaleza situaba genes bacterianos en las bacterias y genes de anfibio en los anfibios, ¿quiénes éramos nosotros para intercambiarlos? ¿No podrían ser terribles las consecuencias? En un congreso celebrado en Asilomar en 1974, se discutieron a fondo las razones de seguridad y se concluyó reanudar prudentemente la Ingeniería Genética en Norteamérica, bajo la supervisión de un Comité Federal. La ciencia se estaba politizando. Poco a poco, la ansiedad pública pareció calmarse, aunque repentinamente se reavivó a mediados de los años noventa; esta vez el centro de atención no era la seguridad, sino la ética [12].

Había nacido la Biotecnología. Primero Genetech, después Cetus y Biogen, y luego otras compañías, aparecieron para explotar la nueva técnica. Un mundo de posibilidades se presentaba ante las empresas nacientes. Ahora se podía inducir a las bacterias a fabricar proteínas para uso médico, alimenticio o industrial. Poco a poco se fue asomando la decepción cuando resultó que, las bacterias no eran muy hábiles fabricando la mayoría de las proteínas humanas y que la demanda de proteínas humanas como medicinas no era demasiado alta. Sin embargo, hubo resultados. Para finales de la década de los ochenta, la hormona humana del crecimiento fabricada por bacterias había sustituido a la extraída del cerebro de los cadáveres, cara y peligrosa [2].

Hasta ahora, los temores éticos y de seguridad resultaron infundados: en treinta años de ingeniería genética no se ha producido ningún accidente medioambiental ni de salud pública derivado de un experimento de ese tipo. Por ahora todo va bien.

Entretanto, la Ingeniería Genética obtuvo un mayor impacto en la ciencia que en el mundo de los negocios. Ya era posible "clonar" genes: aislar una "aguja" de

gen humano del "pajar" que constituye el genoma, ponerlo en una bacteria y reproducir millones de copias suyas, de modo que se puedan purificar y se pueda leer su secuencia de letras. De esa manera se han creado bancos enormes de ADN que contienen miles de fragmentos solapados del genoma humano, presentes en cantidad suficiente para su estudio.

A partir de estos bancos, las personas que hay detrás del Proyecto Genoma Humano han reunido las piezas del texto completo. El proyecto empezó a finales de los años ochenta con el propósito absurdamente ambicioso de leer todo el genoma humano en unos veinte años.

Durante catorce años avanzó poco, entonces, en un sólo año nuevas máquinas secuenciadoras de genes hicieron el trabajo. El 26 de junio del año 2000, el proyecto anunció que había completado un "borrador aproximado" de la receta humana [12].

El cuerpo humano contiene aproximadamente cien billones de células, la mayoría de las cuales tiene un diámetro de menos de una décima de milímetro, dentro de cada célula hay un corpúsculo negro llamado Núcleo, dentro del núcleo se encuentran dos series completas del Genoma humano. Una serie del genoma procede de la madre y otra del padre; en principio, cada serie comprende los mismo treinta mil a ochenta mil genes en veintitrés cromosomas.

En la práctica, existen a menudo pequeñas y sutiles diferencias entre las versiones paterna y materna de cada gen, diferencias que explican, por ejemplo, los ojos azules o castaños. Cuando procreamos, transmitimos una serie completa, pero sólo después de intercambiar fragmentos de los cromosomas paternos y maternos en un proceso conocido como Recombinación “crossover” [1].

De manera breve y concisa podemos definir al genoma humano como todo el conjunto de genes alojados en 23 pares de cromosomas, constituye nada menos que una autobiografía de nuestra especie. El genoma ha sido corregido, abreviado, modificado y aumentado a medida que se ha transmitido de generación en generación a lo largo de más de tres mil millones de años.

3.3 MANIPULACIÓN GENÉTICA APLICADA A UN ALGORITMO GENÉTICO

Ahora veremos cómo aplicar el principio sobre el cual se basa la manipulación genética, para corregir o intercambiar genes defectuosos, a la técnica de los algoritmos genéticos.

En la sección 3.1, se enunciaron las desventajas que podría traer el hecho de conservar relaciones entre genes que perjudiquen a los cromosomas de las futuras generaciones.

Retomando el ejemplo de esa sección tenemos

B D **E A C H** G F → B D **B C F H** G F → E D **B C F H** G A
G A **B C F H** E D → G A **E A C H** E D → G F **E A C H** B D

Figura 3.2 Cruce entre dos cadenas empleando PMX.

Sabemos que la relación que amenaza con incrementar la probabilidad de no encontrar el recorrido más óptimo es la que existe entre **C** y **F**.
Por lo tanto se procede a reorganizar los genes de la siguiente manera:

Una vez que se han realizado los cruces propiciados en esta ocasión, por el operador genético de cruce PMX, tenemos el recorrido descrito por **EDBCFHGA**, que es el individuo o cromosoma dentro del cual se encuentra la relación **CF**, el objetivo es separarlos sin violar las restricciones impuestas por el contexto del Problema del Agente Viajero.

Considerando lo anterior, se hace:

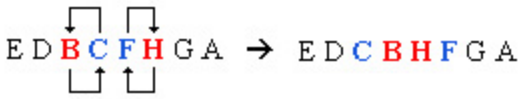


Figura 3.3 Separación de ciudades

La ciudad **C**, que ocupa la cuarta posición en el cromosoma, intercambia su lugar con la ciudad que se encuentra próxima a su izquierda, en este caso **B**. De igual forma la ciudad **F** ubicada en la quinta posición realiza lo mismo pero con la próxima ciudad ubicada a su derecha, o sea **H**.

Se ha modificado el código genético del cromosoma. El nuevo recorrido sigue siendo válido, no hay ciudades que se repitan en él ni tampoco ciudades que hagan falta, es totalmente legal.

Realizado este proceso, el individuo es agregado a la población como miembro de la nueva generación.

Existen dos casos especiales que pueden presentarse.

Caso 1 La "relación amenazadora" se encuentra al inicio del recorrido.

El par de genes a separar se encuentra conformado por las dos primeras ciudades del cromosoma.

C F H E D G A **B** → **B H F** E D G A **C**

Figura 3.4 Pareja ubicada al inicio.

Lo que se hace es lo siguiente. Al igual que antes la ciudad ubicada en la segunda posición del cromosoma intercambia posición con su vecino próximo a la derecha, es decir **F** con **H**. Como la ciudad **C** es la ubicada en la primera posición, no tiene vecino próximo a la izquierda, entonces su posición es intercambiada con la ciudad ubicada en la última posición del cromosoma, en este caso **B**.

Caso 2 La "relación amenazadora" se encuentra al final del recorrido.

El par de genes a separar se encuentra conformado por las dos últimas ciudades del cromosoma.

B E D G A **H C F** → **F** E D G A **C H B**

Figura 3.5 Pareja ubicada al final.

Ahora bien, la situación es la contraria, la penúltima ciudad que es **C**, intercambia posición con **H** que es su vecino próximo a la izquierda, mientras quien no tiene vecino próximo por estar al final del cromosoma es **F**, por lo tanto el cambio se realiza ente la última y la primera ciudad **B**. De esta forma se cubren los diferentes casos en que puede presentarse de acuerdo a su posición, una relación entre dos ciudades, que posiblemente amenace con perjudicar la eficiencia del recorrido.

3.4 TÉCNICA: PROCEDIMIENTO

El proceso de manipulación genética aplicado al Problema del Agente Viajero mediante Algoritmos Genéticos se describe de la siguiente manera:

Paso 1

Analizar el cromosoma del individuo y determinar la relación con el valor más alto (hablamos de kilómetros).

Paso 2

Intercambiar las posiciones de las dos ciudades que forman la relación con sus vecinos próximos de ambos lados.

Nota. En caso de referirnos a la primera ciudad del recorrido, su vecino próximo a la izquierda será la última ciudad del recorrido, de igual forma para la última ciudad del recorrido su vecino próximo a la derecha será la primera ciudad del recorrido.

Paso 3

Realizado el cambio, el cromosoma es vuelto a inspeccionar comparando el antiguo valor de la relación encontrada como la mas alta, con el que surja en esta nueva búsqueda como el mas alto. Si el nuevo valor es más alto, se regresa al paso 2, de lo contrario el procedimiento se da por concluido.

Es importante señalar que esta serie de pasos se debe llevar acabo dentro del procedimiento que manejan los AGs. El AG siembra una población, a partir de la cual comienzan a surgir las siguientes generaciones, mediante el uso de los operadores genéticos de cruce y mutación, aunado a un método de selección para elegir a los individuos padres. El proceso de manipulación genética es aplicado justo antes de que se muestre la nueva generación obtenida; por lo tanto, se puede decir que:

Primero, se seleccionan a los padres que se cruzarán.
Segundo, se realizan los mapeos o cruces obteniendo la nueva generación.
Tercero, de ser necesario se aplica la mutación a los individuos de la nueva generación.
Cuarto, se aplica la manipulación genética a los individuos de la nueva generación.
Quinto, se obtiene la nueva población.

Es importante dejar en claro que la técnica descrita en éste capítulo puede ser generalizada a otro tipo de problemas y no pensar que es solamente aplicable al Problema del Agente Viajero.

En el capítulo 4 se verá la forma en que la Técnica de Manipulación Genética es empleada para resolver el Problema del Agente Viajero, no solo se le dará solución, sino se mostrará las ventajas que la técnica presenta sobre la técnica empleada por un Algoritmo Genético convencional.

CAPÍTULO 4.-
Aplicación a un Caso Real

En este capítulo, se resolverá el problema planteado anteriormente del Agente Viajero para un caso real, mediante un AG convencional y un AG empleando Manipulación Genética.

4.1 PLANTEAMIENTO

El problema consiste en resolver el TSP para diez ciudades ubicadas en la República Mexicana. Las ciudades son: Ciudad Victoria Tamaulipas; Monterrey, Nuevo León; Durango, Durango; Chihuahua, Chihuahua; Culiacán, Sinaloa; Guanajuato, Guanajuato; Chilpancingo, Guerrero; Veracruz, Veracruz; Campeche, Campeche; y Tuxtla Gutierrez, Chiapas. Cuya ubicación es presentada en la siguiente figura:

Figura 4.1 Ubicación Geográfica de cada una de las ciudades.

Las ciudades se etiquetan como se indica a continuación:

Ciudad	Etiqueta
Cd.Victoria, Tamaulipas.	1
Monterrey, Nuevo León.	2
Durango.	3
Chihuahua.	4
Culiacán Sinaloa.	5
Guanajuato.	6
Chilpancingo, Guerrero.	7
Veracruz.	8
Campeche.	9
Tuxtla Gutierrez, Chiapas.	10

Tabla 4.1 Simbología asignada a cada ciudad para el caso real.

La distancia comprendida entre cada una de las ciudades se muestra en la tabla siguiente:

	Cd. Vic	Mty	Dgo	Chih	Cul	Gto	Chilp	Ver	Camp	Tuxt
Cd. Vic, Tamps.	0									
Monterrey	287	0								
Durango	822	615	0							
Chihuahua.	1131	818	709	0						
Culiacán	1361	1154	539	1248	0					
Guanajuato	4804	662	601	1143	1033	0				
Chilpancingo	975	1271	1295	1747	1584	633	0			
Veracruz	749	1036	1327	1854	1797	866	716	0		
Campeche	1610	1897	2227	2497	2610	1677	1572	861	0	
Tuxtla, Chis.	1522	1809	1921	2473	2391	1460	1089	773	674	0

Tabla 4.2 Relación de distancias entre ciudades para el caso real.

4.2 SOLUCIÓN EMPLEANDO UN ALGORITMO GENÉTICO CONVENCIONAL

Para dar inicio a la solución del TSP mediante un AG, se proponen cuatro recorridos los que sean, para formar la población inicial de arranque. Como se indicó en el capítulo 1, la población inicial debe estar constituida por individuos distintos entre sí, esto quiere decir recorridos diferentes.

Se proponen los siguientes:

Población Inicial
10 – 1 – 9 – 8 – 2 – 3 – 7 – 6 – 4 – 5 km = 12354
3 – 8 – 1 – 6 – 9 – 4 – 10 – 7 – 2 – 5 km = 17580
5 – 6 – 7 – 2 – 8 – 1 – 3 – 4 – 10 – 9 km = 12010
4 – 3 – 7 – 5 – 8 – 2 – 9 – 10 – 6 – 1 km = 16387

Tabla 4.3 Población de Inicio.

Cada uno de ellos con el total de kilómetros que implica recorrerlos.

Ahora se seleccionan de manera aleatoria cuatro de ellos, sin importar que pudiera salir seleccionado más de una vez cualquiera de los cuatro. La forma en que se lleva acabo la selección es mediante el método de la "Ruleta Sesgada" [14][Anexo A]. Respetando el orden en que son seleccionados, se forman dos parejas. Se aplica uno de los operadores genéticos de cruce vistos en el capítulo 2, y de cada una de las parejas se obtienen dos recorridos nuevos, llamados recorridos-hijos, que serán los integrantes de la nueva población. De esta manera se forma la primera generación. El operador genético de mutación es empleado cada determinado número de generaciones para evitar caer en ciclamientos. En esta ocasión cada 4 generaciones es aplicado.

Para aclarar más esto, analicemos el paso de la población inicial a la primera generación, del caso que se está resolviendo.

Los cuatro recorridos seleccionados fueron:

Pareja 1

5 - 6 - 7 - 2 - 8 - 1 - 3 - 4 - 10 - 9
5 - 6 - 7 - 2 - 8 - 1 - 3 - 4 - 10 - 9

Pareja 2

10 - 1 - 9 - 8 - 2 - 3 - 7 - 6 - 4 - 5
5 - 6 - 7 - 2 - 8 - 1 - 3 - 4 - 10 - 9

El operador de cruce empleado es el OX, "Order Crossover", (ver capítulo 2). Los puntos de cruce son el 4 y el 7. Los puntos de cruce son generados de manera aleatoria. Aplicando los cruces correspondientes se tiene la nueva población que constituye a la primera generación.

Primera Generación
5 – 6 - 7 – 2 - 8 – 1 - 3 – 4 - 10 - 9 km = 12010
5 – 6 - 7 – 2 - 8 – 1 - 3 – 4 - 10 - 9 km = 12010
5 – 6 - 1 – 8 - 2 – 3 - 7 – 4 - 10 - 9 km = 17036
10 - 9 - 7 - 2 - 8 - 1 – 3 - 6 - 4 - 5 km = 11507

Tabla 4.4 Primera Generación obtenida.

De la misma forma en que se obtuvo la primera generación, se obtiene la segunda solo que ahora se toma como población inicial a la que constituye a la primera generación. Se puede obtener el número de generaciones que uno desee, no hay límites.

Para un problema no lineal de maximización con dos variables resuelto por AGs, se realizaron 1000 iteraciones (generaciones), dándose cuenta que el mejor cromosoma o solución óptima se encontraba en la generación 419 [6].

4.2.1 RESULTADOS OBTENIDOS

Antes de dar paso a los resultados que se obtuvieron, resolviendo el problema del TSP empleando un AG con el operador de cruce OX, es importante mencionar y considerar lo siguiente.

El TSP es un problema más complejo de lo que aparenta, a pesar de que su descripción no lleva mas de cuatro o cinco líneas, su laborioso proceso de cálculo crece de manera exponencial a medida que el número de elementos (ciudades) involucrados aumenta. Recordemos que para calcular la cantidad total de recorridos que pueden formarse para n ciudades, simplemente se calcula su factorial, es decir $n!$. Así, si hablamos de un problema con 5 ciudades, nos referimos a buscar nuestra solución dentro de 120 recorridos, si es un problema de 6 ciudades entonces la solución se encontrará dentro de 720 recorridos. Como puede verse el número total de recorridos creció de 5 a 6 ciudades, pero no tanto como el de 6 a 7, que es 5,040 recorridos.

Es claro que a medida que aumenta el número de ciudades, el total de recorridos se dispara enormemente, sino veamos el caso para 10 ciudades, que es la cantidad que se está manejando en este ejemplo. El factorial de 10 es 3,628,800.

Una forma de resolver el TSP es hacerlo mediante búsqueda exhaustiva, pero aún con ayuda de una computadora convencional el tiempo de cálculo sería algo tardado. Para calcular los 3,628,800 recorridos con una computadora Pentium III a 766 MHz, se llevo dos horas de tiempo. El problema se programó en lenguaje C, Borlandc 3.1 y se pudo observar que el recorrido más corto es de 7,392 kilómetros, algunos de los recorridos encontrados con este valor son

9 10 7 6 5 3 4 2 1 8 km = 7392
2 4 3 5 6 7 10 9 8 1 km = 7392
1 2 4 3 5 6 7 10 9 8 km = 7392

Se puede observar que la relación entre las ciudades es la misma en cada recorrido, simplemente que se encuentran en distinta posición dentro del cromosoma. Dicho de otra forma, los tres recorridos son el mismo, solo que la ciudad tomada como punto de salida es la que varía, por lo tanto como estamos trabajando con 10 ciudades existen entonces 10 recorridos con distinto punto de salida.
Otro punto importante también a considerar, es que puede pensarse que a cada recorrido calculado le corresponde un único kilometraje, es decir si el recorrido ABCDE implica recorrer 500 kilómetros, no quiere decir que ningún otro

recorrido diferente no pueda implicarlos. Quizás en el recorrido CAEBD también la suma de las distancias entre sus ciudades, sume 500 kilómetros. No es una relación uno a uno, es una relación muchos a uno [6].

En esta ocasión, con el AG convencional se calcularon 500 generaciones, considerando que eran suficientes para encontrar a la ruta más óptima. Auxiliados por la computadora para realizar los cálculos el AG fue programado en lenguaje C, Borlandc 3.1. El programa fue diseñado de tal forma que la salida se manda a escribir en un archivo de texto. Las 500 generaciones son escritas, descritas y mostradas en este archivo. Parte del contenido del archivo se muestra a continuación:

```
POBLACION INICIAL
Padre1 : 10 1 9 8 2 3 7 6 4 5 km=12354
Padre2 : 3 8 1 6 9 4 10 7 2 5 km=17580
Padre3 : 5 6 7 2 8 1 3 4 10 9 km=12010
Padre4 : 4 3 7 5 8 2 9 10 6 1 km=16387
0.212, 0.513, 0.719, 1.000  4-7  0.990 0.950 0.410 0.780
GENERACION 1
Padre1 : 5 6 7 2 8 1 3 4 10 9 km=12010
Padre2 : 5 6 7 2 8 1 3 4 10 9 km=12010
Padre3 : 5 6 1 8 2 3 7 4 10 9 km=17036
Padre4 : 10 9 7 2 8 1 3 6 4 5 km=11507
0.228, 0.457, 0.781, 1.000  7-8  0.760 0.720 0.800 0.330
GENERACION 2
Padre1 : 5 6 7 2 8 1 3 4 10 9 km=12010
Padre2 : 5 6 7 2 8 1 3 4 10 9 km=12010
Padre3 : 5 6 2 8 1 3 7 4 10 9 km=13101
Padre4 : 5 6 1 8 2 7 3 4 10 9 km=16654
0.223, 0.447, 0.690, 1.000  6-9  0.370 0.900 0.430 0.040
```

Figura 4.2 Fragmento del Archivo de Salida

Es sólo un pequeño fragmento del archivo, es el inicio, son las primeras 2 generaciones de 500. El archivo además de mostrar los recorridos de cada generación, muestra la información requerida para generar los recorridos de la siguiente generación. Esta información se encuentra ubicada al final de cada generación.

Por ejemplo, al final de la primera generación, las primeras cuatro cifras que aparecen (0.228, 0.457, 0.781, 1.000) de izquierda a derecha, es la probabilidad acumulada de cada individuo de ser seleccionado, las últimas cuatro cifras (0.760, 0.720, 0.800, 0.330) son valores entre 0 y 1 generados de manera aleatoria, son empleados junto con las probabilidades acumuladas por el método de la “Ruleta Sesgada” [14] para seleccionar a los individuos que formarán las dos parejas, y la expresión (7-8) indica los puntos de cruce que se aplican a los individuos que resulten de la selección. Con esto y una vez aplicado el operador de cruce se obtienen los cuatro integrantes de la segunda generación.

En otro archivo son recolectados únicamente los kilómetros totales de cada uno de los recorridos. Los valores fueron graficados en el paquete computacional de cálculo llamado Excel y se obtuvo la siguiente gráfica:

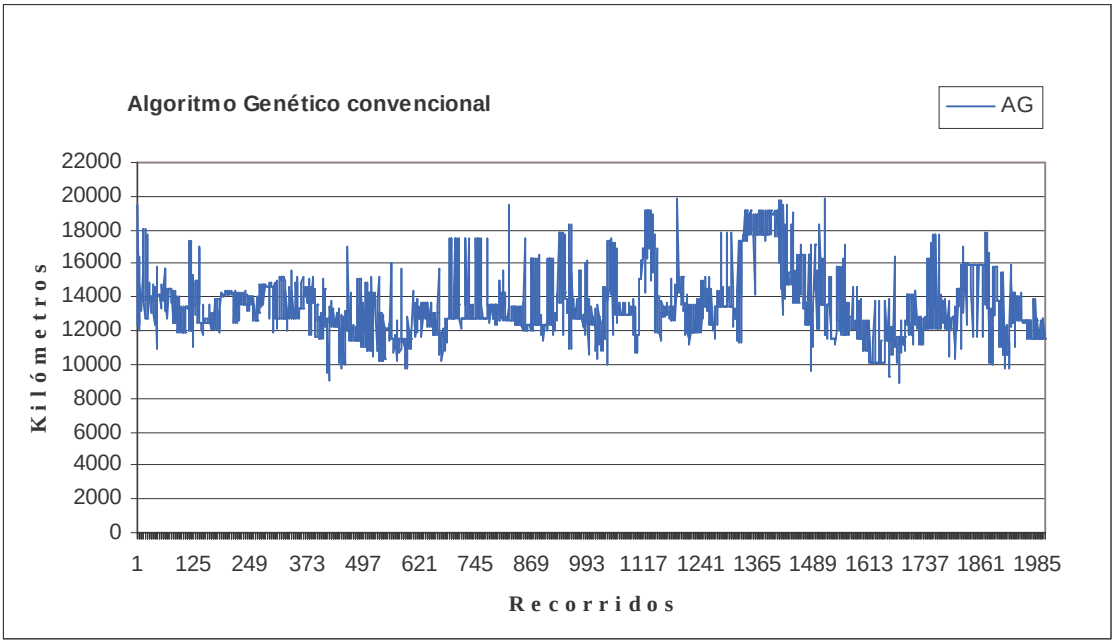


Figura 4.3 Gráfica de la salida calculada por el AG convencional.

El valor más pequeño que se encuentra en la gráfica es de 8,914 kilómetros, correspondiente al individuo 1678, localizados en la generación 420. (Si multiplico el número de generación por el tamaño de la población, obtengo el número que le corresponde al último individuo de esa generación, de esta forma ubico a los individuos restantes).

```
MUTACION-GENERACION 420
Padre1 : 6 8 7 5 4 3 1 10 2 9 km=12850
Padre2 : 10 8 7 5 4 3 1 2 6 9 km=8914
Padre3 : 6 8 7 5 4 3 1 2 10 9 km=10392
Padre4 : 6 8 7 5 4 3 1 2 10 9 km=10392
0.300, 0.543, 0.786, 1.000  2-5  0.660 0.200 0.750 0.120
```

Figura 4.4 Generación donde se encuentra el recorrido más corto.

El título de “Mutación-Generación” se emplea para indicar que para obtener esa generación fue aplicado el operador genético de mutación (ver capítulo 2, Intercambio Recíproco).

El recorrido encontrado por el AG como el más óptimo, es el que se muestra en el mapa.

Figura 4.5 Recorrido obtenido por el AG convencional.

4.3 SOLUCIÓN EMPLEANDO MANIPULACIÓN GENÉTICA

Se realiza el mismo procedimiento empleado en el AG convencional. Las generaciones son obtenidas exactamente del mismo modo, el operador de cruce puede ser cualquiera, en esta ocasión con fines de comparación se empleó el mismo usado en el AG convencional. El operador de mutación también es empleado con la misma periodicidad.

La diferencia radica esencialmente en hacer un paso extra. Una vez que se tiene lista la siguiente generación, antes de ser mostrada como tal y considerada para el cálculo de la siguiente, es sometida a una inspección. Cada miembro de la generación es analizado con el fin de encontrar como ya se mencionó y describió en el capítulo 3, relaciones entre genes que perjudiquen al cromosoma o recorrido.

Cuando estas relaciones han sido encontradas, son destruidas como indica el proceso descrito en la sección 3.4.
La finalidad de esto es mejorar el cromosoma. El valor del recorrido en kilómetros de cada individuo después de ser sometido a este proceso, casi siempre es disminuido, mejorando así en promedio las características de la población en general.

4.3.1 RESULTADOS OBTENIDOS POR MEDIO DE LA MANIPULACIÓN GENÉTICA

Al igual que en el AG convencional, se elaboró un programa de cómputo para aplicar la técnica descrita. Se tienen también dos archivos de salidas, uno con todas las generaciones calculadas y los datos necesarios para calcular la siguiente generación, y el otro con valores de los kilómetros de cada uno de los recorridos.

Se realizaron 500 generaciones, tomando la misma población inicial utilizada en el AG convencional. Una vez mas los valores fueron graficados en Excel, y se obtuvo lo siguiente:

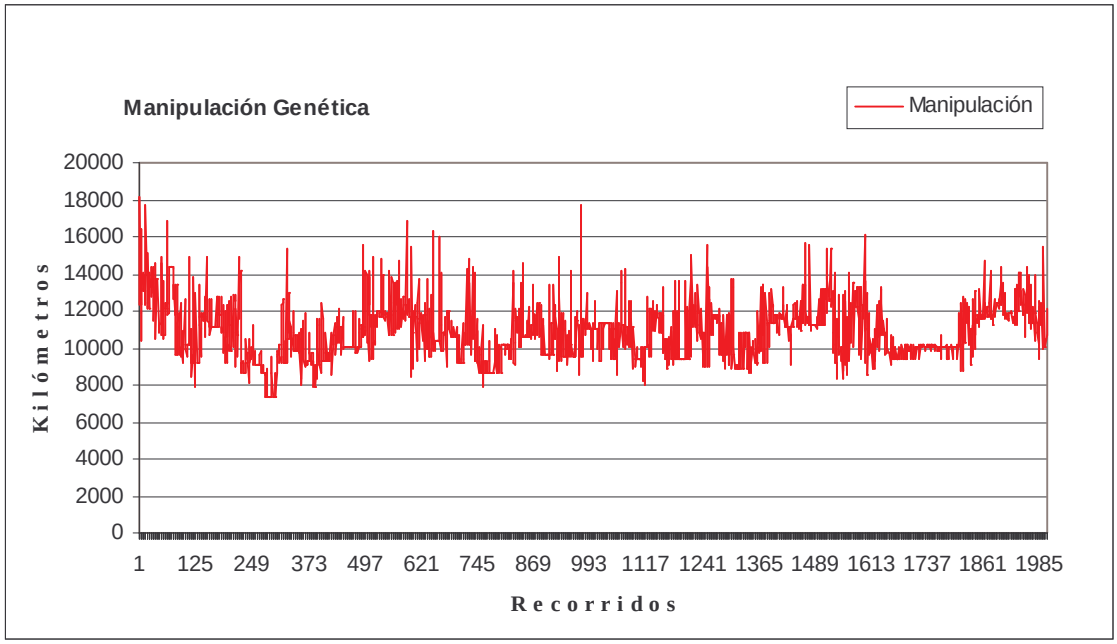


Figura 4.6 Gráfica de la salida calculada por la MG.

El valor más pequeño que se encuentra en la gráfica es de 7,392 kilómetros, correspondiente a los individuos 282, 288, 290, 297, 301 y 302, localizados en las generaciones 71, 72, 73, 75 y 76.

```
GENERACION 71
Padre1 : 10 9 7 3 2 5 1 4 6 8  km=10584
Padre2 : 9 10 7 6 5 3 4 2 1 8  km=7392
Padre3 : 2 10 7 6 5 3 4 9 1 8  km=11704
Padre4 : 10 9 7 3 2 5 1 4 6 8  km=10584
0.294, 0.500, 0.706, 1.000  2-8  0.820 0.330 0.430 0.850
MUTACION-GENERACION 72
Padre1 : 6 9 7 3 2 5 4 1 8 10  km=11674
Padre2 : 10 9 7 3 2 5 1 4 6 8  km=10584
Padre3 : 6 9 7 3 2 5 1 4 10 8  km=12917
Padre4 : 9 10 7 6 5 3 4 2 1 8  km=7392
0.174, 0.422, 0.725, 1.000  1-9  0.780 0.920 0.510 0.380
GENERACION 73
```

```
Padre1 : 6 9 7 3 2 5 4 1 8 10  km=11674
Padre2 : 9 10 7 6 5 3 4 2 1 8  km=7392
Padre3 : 6 9 7 3 2 5 4 1 8 10  km=11674
Padre4 : 10 9 7 3 2 5 1 4 6 8  km=10584
0.245, 0.490, 0.746, 1.000  6-8  0.950 0.160 0.510 0.970
GENERACION 75
Padre1 : 2 4 3 5 6 7 10 9 8 1  km=7392
Padre2 : 6 3 4 5 7 9 10 1 2 8  km=10099
Padre3 : 8 5 3 4 6 10 7 9 2 1  km=11242
Padre4 : 2 5 3 4 6 10 7 9 8 1  km=9563
0.264, 0.457, 0.750, 1.000  2-6  0.210 0.230 0.940 0.350
MUTACION-GENERACION 76
Padre1 : 2 4 3 5 6 7 10 9 8 1  km=7392
Padre2 : 2 4 3 5 6 7 10 9 8 1  km=7392
Padre3 : 4 2 3 5 6 7 8 9 10 1  km=8542
Padre4 : 8 5 3 4 6 7 10 9 2 1  km=9517
0.314, 0.544, 0.744, 1.000  8-9  0.610 0.340 0.630 0.070
```

Figura 4.7 Generaciones donde se encuentra el recorrido más corto.

El recorrido encontrado por el AG con manipulación genética como el más óptimo, es el que se muestra en el mapa.

Figura 4.8 Recorrido obtenido con la MG.

4.4 COMPARACIÓN DE RESULTADOS ENTRE UN AG Y LA MG

El problema para el caso real del TSP con 10 ciudades, fue resuelto en un inicio mediante la búsqueda exhaustiva. Con ayuda de un programa se calcularon todas las combinaciones posibles, y se encontró cuál de todas ellas cubría la menor cantidad de kilómetros, que fue de 7,392. Se mencionó lo tardado que es realizar el cálculo total, más aún si el número de ciudades se incrementa. Tan solo decir que para 11 ciudades, serían 39,916,800 recorridos diferentes, aproximadamente 22 horas con una computadora de las características descritas en el capítulo anterior.

Con el método del AG usando el operador de cruce OX, el tiempo de cálculo fue de 30 segundos aproximadamente, para 500 generaciones en la misma computadora. Si se desea ampliar el número de iteraciones no hay problema alguno, el tiempo de proceso se incrementa sólo unos segundos mas, realmente el algoritmo opera rápidamente. Fue probado hasta para 2000 generaciones, lo que significa generar 8000 recorridos.

El resultado que entrega el programa es un recorrido con 8,914 kilómetros. De la misma forma en que sabemos, gracias a la búsqueda exhaustiva que se hizo, que el recorrido más corto es de 7,392 kilómetros, sabemos también que los recorridos más grandes se encuentran arriba de 20,000 kilómetros. Por lo tanto, es una solución muy aproximada y aceptable, pero comparémosla con el otro algoritmo.

En el AG con manipulación genética, el tiempo de cálculo fue prácticamente el mismo, al igual que en el AG convencional si se desea incrementar el número de generaciones calculadas puede hacerse sin preocuparse por el tiempo. Con 500 generaciones el resultado arrojado por el programa fue de 7,392 kilómetros, que son los mismos kilómetros que comprenden el recorrido más corto encontrado en la búsqueda exhaustiva.

En esta ocasión no hubo aproximación, el resultado es exactamente el mismo. El AG convencional fue superado en precisión por el AG con manipulación genética, la convergencia fue más rápida.

Es difícil, dada la esencia de un AG que radica principalmente en simular un comportamiento natural, lleno de probabilidades y alternativas, fijar las mismas condiciones para ambos algoritmos en comparación. Sin embargo, se consideró la misma población de inicio para ambos, la misma cantidad fija de individuos en cada generación (4), el mismo método de selección de individuos padres (Ruleta Sesgada), el mismo operador de cruce (OX) y la misma cantidad de generaciones calculadas (500).

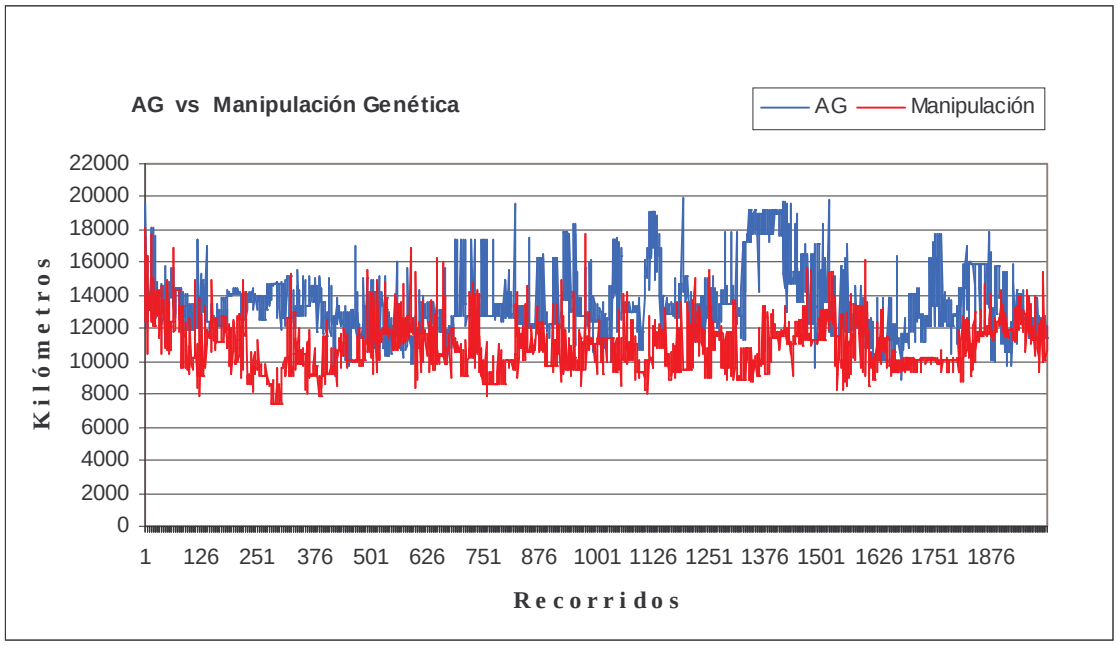


Figura 4.9 Comparación de las salidas obtenidas por ambas técnicas.
Caso para diez ciudades.

En la comparación de gráficas se puede observar que en promedio el AG con manipulación genética, siempre estuvo por debajo del AG convencional.

La siguiente figura es un cuadro comparativo de los resultados obtenidos con ambas técnicas:

	AG	MG
Población inicial	La misma	La misma
Generaciones calculadas	500	500
Generación donde se encontró el óptimo por primera vez	Ninguna	Generación 71
Lenguaje de programación	Borlandc C 3.1	Borlandc 3.1
Tiempo de calculó	Aprox. 30 segundos	Aprox. 30 segundos
Mutación aplicada	Cada 4 generaciones	Cada 4 generaciones
Operador de cruce empleado	OX	OX

Tabla 4.5 Cuadro comparativo entre ambas técnicas para el caso real.

El AG convencional también converge, el programa se ejecutó diez veces más, en algunas ocasiones para 500, 700 y 1000 generaciones, con distintas poblaciones de inicio. Solo en tres ocasiones se alcanzó el valor de 7,392 kilómetros.

4.5 APLICACIÓN AL CASO DE: CINCO Y OCHO CIUDADES

Antes de finalizar este capítulo haremos una comparación más entre ambas técnicas (AG convencional y AG con Manipulación Genética) resolviendo un caso para 5 ciudades y otro para 8.

Primero resolvamos el caso con 5 ciudades. Supongamos 5 ciudades ficticias, las cuales etiquetaremos con las letras del abecedario como sigue:

Ciudad	Etiqueta
Ciudad 1	A
Ciudad 2	B
Ciudad 3	C
Ciudad 4	D
Ciudad 5	E

Tabla 4.6 Simbología asignada a cada ciudad.

A diferencia del caso real resuelto con 10 ciudades, las ciudades son representadas ahora por letras y no por números. Recordemos que se puede emplear cualquier símbolo para representar una ciudad.

La relación de sus distancias es la siguiente.

	A	B	C	D	E
A	0				
B	89	0			
C	215	304	0		
D	302	391	195	0	
E	920	1009	701	755	0

Tabla 4.7 Relación de distancias entre ciudades.

Cómo la cantidad de ciudades no es muy grande o mejor dicho el factorial de 5 no lo es, podemos mostrar los 120 recorridos existentes que resultan al resolver el problema mediante búsqueda exhaustiva.

12345 km=2263	21345 km=2263	31245 km=2151	41235 km=2151	51234 km=2263
12354 km=2151	21354 km=2151	31254 km=2263	41253 km=2296	51243 km=2296
12453 km=2151	21453 km=2151	31452 km=2585	41352 km=2618	51342 km=2730
12435 km=2296	21435 km=2296	31425 km=2618	41325 km=2585	51324 km=2585
12534 km=2296	21534 km=2296	31524 km=2730	41523 km=2730	51423 km=2618
12543 km=2263	21543 km=2263	31542 km=2585	41532 km=2618	51432 km=2730
13245 km=2585	23145 km=2585	32145 km=2151	42135 km=2151	52134 km=2263
13254 km=2585	23154 km=2585	32154 km=2263	42153 km=2296	52143 km=2296

13452 km=2263	23451 km=2263	32451 km=2585	42351 km=2618	52341 km=2730
13425 km=2730	23415 km=2730	32415 km=2618	42315 km=2585	52314 km=2585
13524 km=2618	23514 km=2618	32514 km=2730	42513 km=2730	52413 km=2618
13542 km=2151	23541 km=2151	32541 km=2585	42531 km=2618	52431 km=2730
14235 km=2618	24135 km=2618	34125 km=2296	43125 km=2263	53124 km=2151
14253 km=2618	24153 km=2618	34152 km=2730	43152 km=2730	53142 km=2618
14352 km=2296	24351 km=2296	34251 km=2730	43251 km=2730	53241 km=2618
14325 km=2730	24315 km=2730	34215 km=2296	43215 km=2263	53214 km=2151
14523 km=2585	24513 km=2585	34512 km=2263	43512 km=2296	53412 km=2296
14532 km=2151	24531 km=2151	34521 km=2263	43521 km=2296	53421 km=2296
15234 km=2730	25134 km=2730	35124 km=2296	45123 km=2263	54123 km=2151
15243 km=2730	25143 km=2730	35142 km=2618	45132 km=2585	54132 km=2585
15342 km=2296	25341 km=2296	35241 km=2618	45231 km=2585	54231 km=2585
15324 km=2618	25314 km=2618	35214 km=2296	45213 km=2263	54213 km=2151
15423 km=2585	25413 km=2585	35412 km=2151	45312 km=2151	54312 km=2263
15432 km=2263	25431 km=2263	35421 km=2151	45321 km=2151	54321 km=2263

Tabla 4.8 Total de recorridos existentes para cinco ciudades.

La solución óptima del problema es el recorrido con 2,151 kilómetros. Veamos la figura que muestra la gráfica donde se compara el resultado de ambas técnicas.

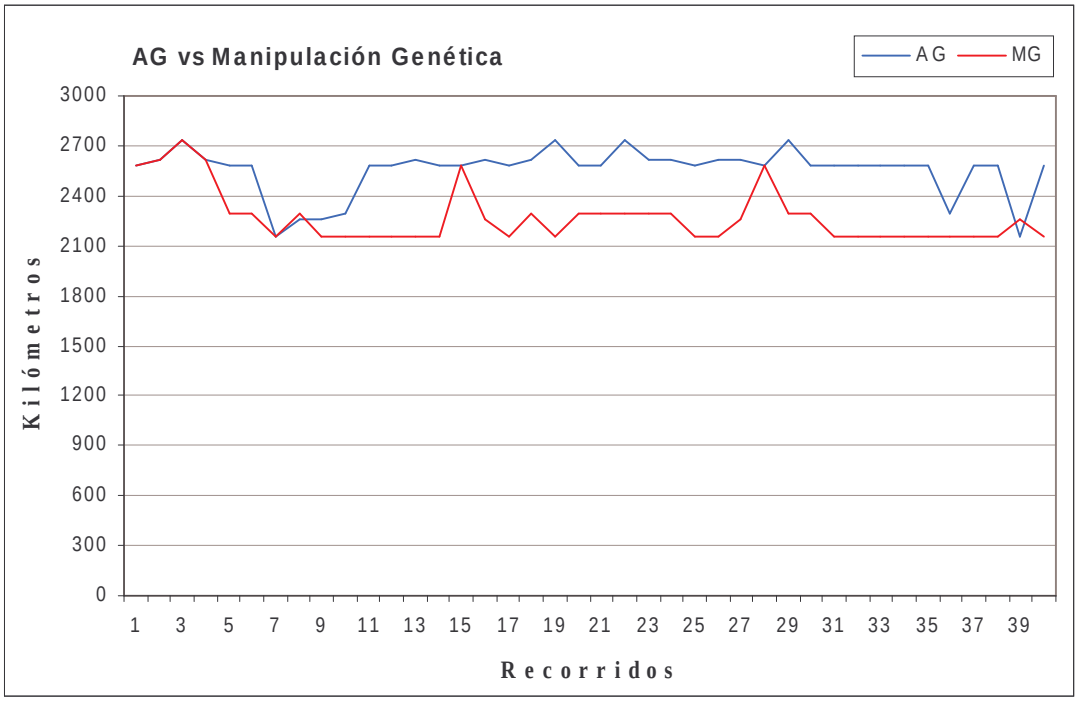


Figura 4.10 Comparación de las salidas obtenidas por ambas técnicas.
Caso para cinco ciudades.

La siguiente tabla es un cuadro comparativo de ambas técnicas en este ejemplo:

	AG	MG
Población inicial	La misma	La misma
Generaciones calculadas	10	10
Generación donde se encontró el óptimo por primera vez	Generación 2	Generación 2
Lenguaje de programación	Borlandc C 3.1	Borlandc 3.1
Tiempo de calculó	Aprox. 3 segundos	Aprox. 3 segundos
Mutación aplicada	Cada 4 generaciones	Cada 4 generaciones
Operador de cruce empleado	OX	OX

Tabla 4.9 Cuadro Comparativo entre ambas técnicas.

Las dos técnicas encontraron el óptimo por primera vez en el tercer individuo de la segunda generación, que corresponde al lugar siete del total de individuos calculados en las diez generaciones, solo que para el AG convencional el recorrido hallado fue ECBAD y para el AG con Manipulación Genética fue BACED. Se puede observar que ambos recorridos mantienen la misma relación de distancias entre sus ciudades.

Ahora veamos el caso para 8 ciudades:

De igual forma las ciudades son ficticias y serán representadas por letras del abecedario.

Ciudad	Etiqueta
Ciudad 1	A
Ciudad 2	B
Ciudad 3	C
Ciudad 4	D
Ciudad 5	E
Ciudad 6	F
Ciudad 7	G
Ciudad 8	H

Tabla 4.10 Simbología asignada a cada ciudad.

La relación de distancias entre las ciudades es:

	A	B	C	D	E	F	G	H
A	0							
B	89	0						
C	215	304	0					
D	302	391	195	0				
E	920	1009	701	755	0			
F	1261	1350	1075	983	533	0		
G	1400	1489	1214	1122	672	139	0	
H	4309	4398	4123	4031	3581	3056	2917	0

Tabla 4.11 Relación de distancias entre ciudades

En este caso, si se quisiera mostrar el total de recorridos que es de 40,320 como se hizo para el ejemplo con 5 ciudades, se ocuparían aproximadamente 160 páginas si en cada una de ellas se escribieran 250 recorridos distribuidos en 5 columnas. Sin embargo, con ayuda del programa que realiza la búsqueda exhaustiva se encontró que el recorrido más corto es de 9,016 kilómetros.

En la siguiente figura, se muestra la gráfica comparativa de los resultados obtenidos con ambas técnicas:

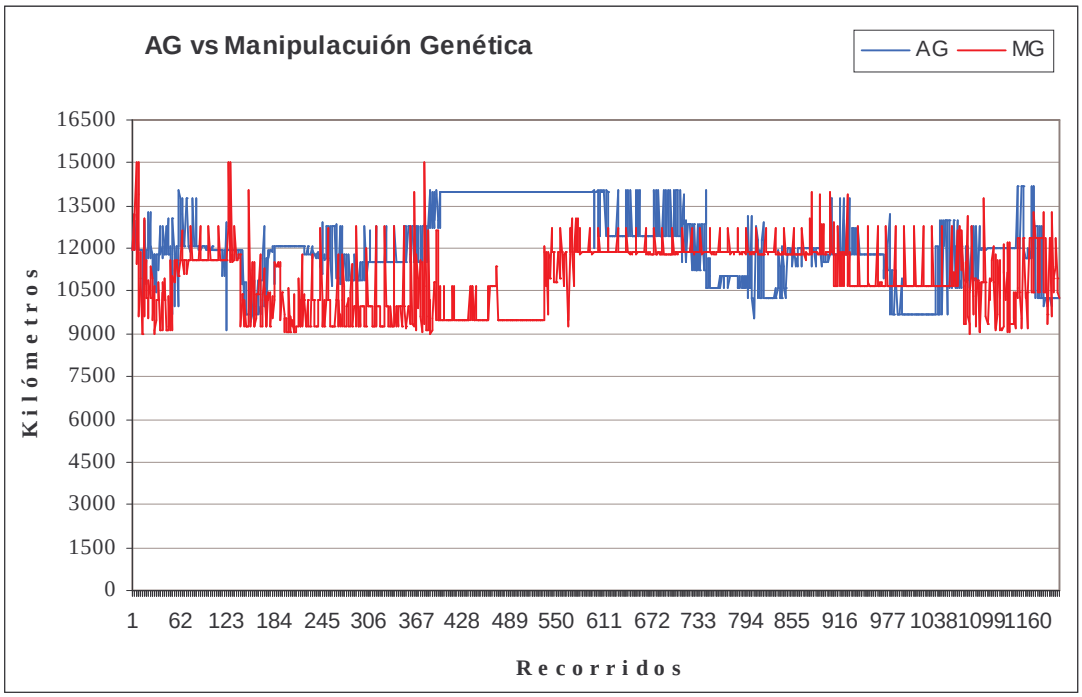


Figura 4.11 Comparación de las salidas obtenidas por ambas técnicas.
Caso para ocho ciudades.

La tabla de abajo es el cuadro comparativo para ambas técnicas en este ejemplo:

	AG	MG
Población inicial	La misma	La misma
Generaciones calculadas	300	300
Generación donde se encontró el óptimo por primera vez	Ninguna	Generación 4
Lenguaje de programación	Borland C 3.1	Borland C 3.1
Tiempo de calculó	Aprox. 10 segundos	Aprox. 8 segundos
Mutación aplicada	Cada 4 generaciones	Cada 4 generaciones
Operador de cruce empleado	OX	OX

Tabla 4.12 Cuadro Comparativo entre ambas técnicas.

Como se puede observar en la gráfica y en el cuadro comparativo, el AG con Manipulación Genética supero al AG convencional ya que este último no encontró el óptimo durante las 300 generaciones, sin embargo el recorrido más bajo que encontró fue de 9,128 kilómetros, que se ubica en la generación 31 y es muy cercano al valor del óptimo.

El primer óptimo hallado por el AG con Manipulación Genética se encuentra en el segundo individuo de la cuarta generación, es decir el que ocupa el lugar catorce en las 300 generaciones y es el recorrido ECABDFGH.
El principal objetivo de éste capítulo, no sólo fue mostrar la manera en que funciona la técnica propuesta llamada “Manipulación Genética”, sino la manera en que aventaja a la técnica aplicada por los AG’s convencionales. Si bien, sólo se resolvieron problemas con diez, ocho y cinco ciudades, no hay duda alguna que la técnica puede ponerse a prueba con más ciudades extendiendo así su aplicación, tomando siempre el mismo principio sobre el cual se basa la técnica, mostrado en el capítulo 3.

Con el siguiente capítulo se concluye éste trabajo de tesis, en él se encuentran los comentarios y sugerencias hacia el mismo.

CAPÍTULO 5
Conclusiones

5.1 CONCLUSIONES Y RECOMENDACIONES

El AG funciona y funciona correctamente, ha sido empleado desde hace ya varias décadas. Se mostró que el problema del TSP propuesto con diez ciudades y sus distancias reales, fue resuelto correctamente empleando la técnica de manipulación genética.

Durante mi trabajo de investigación pude observar y determinar, que para problemas con un número pequeño de ciudades ambos algoritmos (AG convencional y AG con manipulación genética), funcionan de igual manera, no hay diferencia. El problema para la convergencia se torna más complicado a medida que crece el número de ciudades y es ahí donde el algoritmo con la técnica propuesta fue más eficiente. Esta comparación se pudo llevar acabo gracias a que fue diseñado un programa computacional para imitar el comportamiento de cada algoritmo y dar solución al problema del TSP.

El propósito de este trabajo de tesis no es desplazar a los métodos existentes, sino de ofrecer esta técnica como otra alternativa para dar solución a problemas de este tipo.

Mis recomendaciones y sugerencias para quien deseé retomar este trabajo, es la de llevar la misma técnica a otros problemas, sobre todo a aquellos donde la codificación es diferente a la del TSP; por ejemplo, en el problema de Goldberg [17], se trabaja con una codificación basada en el código Gray (ceros y unos). En primera instancia me parece interesante, identificar lo que llamo yo en el TSP una “relación peligrosa”. En el TSP, existe una relación de cada gen con cada uno de los demás genes, esa relación es la distancia que hay entre una y otra ciudad. En problemas como el de Goldberg no existe esa relación. No hay distinción a simple vista entre los genes que se encuentran dentro del cromosoma, o es un cero o es un uno.

El principio para aplicar la técnica debe ser el mismo, analizar el cromosoma en busca de un gen o conjunto de genes que afecte la calidad de mi población dependiendo de lo que quiero lograr con ella, y reemplazar esa parte por una nueva, cuidando que no altere la función y desempeño del individuo.

Aparte de Goldberg existen muchos otros problemas que manejan ceros y unos en su codificación.

Otra recomendación es crear un AG en donde se apliquen todos los operadores genéticos de cruce que se mencionaron en el capítulo 2; es decir, que en algunas generaciones se aplique alguno de ellos, en otras otro operador y así sucesivamente. Sería interesante ver los resultados de una combinación de todos los operadores junto con la técnica de manipulación genética. También, se podría diseñar un AG que no ocupe el operador genético de cruce para generar las generaciones y que solamente se base en la técnica de manipulación genética.

El tema de los AGs es un muy amplio se encuentra en constante crecimiento día a día. Sólo por el hecho de estar basados en el principio de la selección natural, permiten que métodos innovadores descubiertos por diferentes disciplinas biología, medicina, ingeniería genética, puedan ser experimentados con ellos y fusionados para mejorar sus resultados y alcances.

Referencias Bibliográficas:

[1] Capecchi, M. R. ALTERING THE GENOME BY HOMOLOGOUS RECOMBINATION, Sience 244, 1989.

[2] Cook-Deegan, R., THE GENE WARS: SCIENCE, POLITICS AND THE HUMAN GENOME, Nueva York, Norton.

[3] Coley, David A. AN INTRODUCTION TO GENETIC ALGORITHMS FOR SCIENTISTS AND ENGINEERS. University of Exeter, 1999.

[4] Davis, L., JOB SHOP SCHEDULING WITH GENETIC ALGORITHM, in Schaffer.

[5] De Jong, K. A. and Sarma J., FOUNDATIONS OF GENETIC ALGORITHMS 2, Morgan Kaufmann, 1993.
http://cs.gmu.edu/faculty/dejong.html

[6] Gen, Mitsuo and Cheng Runwei, GENETIC ALGORITHMS AND ENGINEERING OPTIMIZATION, Ashikaga Institute of Technology Ashikaga, Japan, 2000.

[7] Goldberg, D. A., ALGORITHMS IN SEARCH, OPTIMISATION AND MACHINE LEARNING, Addson-Wesley, 1989.

[8] Goldberg, D. and R. Lingle Alleles, LOCI AND TRAVELING SALESMAN PROBLEM, in Grefenstette.

[9] Kobayashi, S., I. Ono, and M. Yamamura, AN EFFICENT GENETIC ALGORITHM FOR JOB SHOP SCHEDULING PROBLEMS, in Eshelman.

[10] Oliver, I., D. Smith, and J. Holland, A STUDY OF PERMUTATION CROSSOVER OPERATORS ON THE TRAVELING SALESMAN PROBLEM, in Grefenstette.

[11] Princeton University, TRAVELING SALESMAN PROBLEM HOME PAGE www.math.princeton.edu/tsp May 2002 E.U.

[12] Ridley Matt, GENOMA LA AUTOBIOGRAFIA DE UNA ESPECIE EN 23 CAPÍTULOS, Taurus 2001.

[13] Syswerda, G., UNIFORM CROSSOVER IN GENETIC ALGORITHMS, in Schaffer.

[17] Departamento de Ciencias de la Computación e Inteligencia Artificial. Universidad del País Vasco –Euskal Erico Unibertsitatea- ALGORITMOS GENÉTICOS, 1999.

Referencias de Internet:

1.- [14] www.cs.us.es/~delia/sia/html98-99/pag-alumnos/web11/indice.html

2.- [15] www.redcientifica/compevolutive/tsp.html

3.- [16] www.redcientifica/evolution/natural/h_d.html

INDICE DE FIGURAS

INDICE DE TABLAS

Anexo A.- Ruleta Sesgada

En la mayoría de los casos, el modelo de Ruleta Sesgada es adoptado como proceso de selección.

Esta técnica ofrece una adecuada distribución de probabilidad a partir de la cual puede ser seleccionada la nueva población basada en los valores obtenidos.

El proceso de la Ruleta Sesgada es contraído como sigue:

1.- Calcular el valor de la función de evaluación f(*x*) para cada cromosoma:

$$\text{eval}(\text{genotipo}_k) = f(x) \qquad k = 1, 2, \ldots, \textit{tope}$$

*tope = total de individuos en la población.

2.- Calcular el total acumulado de genotipos evaluados de la población:

$$F = \sum_{k=1}^{\text{tope}} \text{eval}(\text{genotipo}_k)$$

3.- Calcular la probabilidad de selección para cada cromosoma:

$$p_k = \frac{\text{eval}(\text{genotipo}_k)}{F}, \qquad k = 1, 2, \ldots, \textit{tope}$$

4.- Calcular la probabilidad acumulada q_k, para cada cromosoma:

$$q_k = \sum_{j=1}^{k} p_j, \qquad k = 1, 2, \ldots, \textit{tope}$$

El proceso de selección comienza girando la ruleta *"tope"* número de veces, en cada ocasión, un cromosoma es seleccionado para formar parte de la nueva población. Para ello es suficiente, con obtener cuatro números reales provenientes de una distribución de probabilidad uniforme en el intervalo (0, 1), es decir generados de manera aleatoria, y compararlos con las probabilidades acumuladas calculadas.

Así por ejemplo, supongamos que trabajamos con una población de cuatro individuos y que sus probabilidades acumuladas son,

$p_1 = 0.14$
$p_2 = 0.63$
$p_3 = 0.69$
$p_4 = 1.0$

y que los cuatro números generados de manera aleatoria hayan sido: 0.58, 0.84, 0.11 y 0.43.

Números aleatorios	Individuo seleccionado
0.63>0.58>0.14	2
1.0>0.84>0.69	4
0.14>0.11>0	1
0.63>0.43>0.14	2

Lo que se hizo fue buscar la ubicación de cada uno de los números aleatorios dentro de las probabilidades acumuladas para identificar cual individuo de la población era seleccionado para formar parte de las parejas. Esto significa que los individuos seleccionados para el cruce han sido: el individuo 2 junto con el individuo 4, así como el individuo 1 junto con el individuo 2.

Anexo B.- Listado del Programa “manipula.c”

En este anexo se encuentra el código fuente del programa en el cual se implementó la técnica de la “Manipulación Genética”. El archivo se llama “manipula.c”.

```
#include<conio.h>
#include<stdio.h>
#include<math.h>
#include<stdlib.h>

typedef struct AG{
  int ciudad, posicion;
  AG *apsig, *apant;
} TSP;

typedef struct distancias{
  int C1, C2, kms;
  distancias *apsig;
} KM;

KM *apdist, *apdistini;
TSP *apnodo1, *apnodoini1, *apnodo2, *apnodoini2, *apnodo3,
*apnodoini3, *apnodo4, *apnodoini4;
FILE *stream1, *stream2;

int crear_nodo(int, int, int, int); //FUNCIÓN PARA CREAR LOS
RECORRIDOS
void cruce1(int, int, TSP *, TSP *); //FUNCIÓN QUE REALIZA EL CRUCE
ENTRE LOS
                                        INDIVIDUOS SELECCIONADOS
void imprime(TSP *, int); //FUNCIÓN QUE IMPRIME LOS RECORRIDOS
void mutacion(TSP *, TSP *); //FUNCIÓN QUE REALIZA LA MUTACIÓN SOBRE
EL
                                  INDIVIDUO SELECCIONADO
int crear_km(int, int, int, int); //FUNCIÓN QUE CREA LA RELACIÓN DE
KILÓMETROS
                                      ENTRE CIUDADES
void recorrido(TSP *, int); //FUNCIÓN EMPLEADA PARA RECORRER CIUDAD
POR CIUDAD
                                  UN RECORRIDO
int distancia(int, int);  //FUNCIÓN QUE CALCULA LA DISTANCIA TOTAL DE
UN
                                RECORRIDO
void porcentaje(long, int); //FUNCIÓN QUE CALCULA LA PROBABILIDAD
ACUMULADA DE
                                  CADA RECORRIDO
float aleatorio(); //FUNCIÓN QUE GENERA UN NÚMERO ALEATORIO
void Genoma(TSP *, int, int, int); //FUNCIÓN QUE APLICA LA TÉCNICA DE
                                       MANIPULACIÓN GENÉTICA

int num=0;
float arr[4];
int aplica=0, again=0, D;

void main()
{
```

```
int bandera=1, i, pos, flag=1, m1=0, m2=0, mx=0, marca, cont=0, k,
kmtotal, indica=0, ar1[10], ar2[10], ar3[10], ar4[10], ar5[10],
ar6[10], m, s=0, mut1a, mut1b, mut2a, mut2b, km1, km2, km3, km4, p,
j=0, arrx[4];
char ar[6], a;
float al;
long suma=0;
TSP *pad1, *pad2, *pad3, *pad4;
clrscr();
      printf("cuántas ciudades tiene el recorrido? "); //TAMAÑO DE LA
POBLACIÓN
      scanf("%d", &num);
      printf("Padre 1:\n");
      for(pos=1; pos<=num; pos++)
      {
            scanf("%d", &i);
            crear_nodo(i, pos, bandera, flag); //SE CREA EL PRIMER
RECORRIDO
            flag=0;
      }
      bandera=2;
      flag=1;
      printf("Padre 2:\n");
      for(pos=1; pos<=num; pos++)
      {
            scanf("%d", &i);
            crear_nodo(i, pos, bandera, flag); //SE CREA EL SEGUNDO
RECORRIDO
            flag=0;
      }
      bandera=3;
      flag=1;
      printf("Padre 3:\n");
      for(pos=1; pos<=num; pos++)
      {
            scanf("%d", &i);
            crear_nodo(i, pos, bandera, flag); //SE CREA EL TERCER
RECORRIDO

            flag=0;
      }
      bandera=4;
      flag=1;
      printf("Padre 4:\n");
      for(pos=1; pos<=num; pos++)
      {
            scanf("%d", &i);
            crear_nodo(i, pos, bandera, flag); //SE CREA EL CUARTO
RECORRIDO

            flag=0;
      }
      clrscr();

//CAPTURA EN KILÓMETROS DE LAS DISTANCIAS ENTRE CIUDADES LEIDAS DESDE
EL ARCHIVO //DISTANCIAS10.FIL
printf ("Inserta la distancia entre las siguientes ciudades\n");
flag=1;
i=2;
k=1;
if ((stream1 = fopen("DISTANCIAS10.FIL", "r"))
```

```
          == NULL)
   {
        fprintf(stderr, "Cannot open input file.\n");
        exit(1);
   }
while(cont < num-1)
{
for(; i<=num; i++)
{
      while(1)
      {
      a=fgetc(stream1);
      ar[j++]=a;
      if(a=='\n')
      {
            p=atoi(ar);
            j=0;
            crear_km(k, i, p, flag);
            flag=0;
            break;
      }
      }
}
k++;
i=k;
i=i+1;
cont ++;
}
fclose(stream1);
clrscr();
randomize();
stream1 = fopen("OX.FIL", "w+");
stream2 = fopen("OXKM.FIL", "w+");
      for(m=0; m<500; m++)
      {
      if(s==3)
      {
            fprintf(stream1, "MUTACION-");
            s=0;
      }
      //SE MUESTRA EN PANTALLA LA POBLACIÓN INICIAL
      fprintf(stream1,"GENERACION %d\n", m+1);
      marca=1;
      recorrido(apnodoini1, num);
      imprime(apnodoini1, marca);
      fprintf(stream1," km=%d\n",D);
      fprintf(stream2,"%d\n",D);
      suma=suma+D;
      marca=2;
      recorrido(apnodoini2, num);
      imprime(apnodoini2, marca);
      fprintf(stream1," km=%d\n",D);
      fprintf(stream2,"%d\n",D);
      suma=suma+D;
      marca=3;
      recorrido(apnodoini3, num);
      imprime(apnodoini3, marca);
      fprintf(stream1," km=%d\n",D);
      fprintf(stream2,"%d\n",D);
      suma=suma+D;
      marca=4;
```

```
	recorrido(apnodoini4, num);
	imprime(apnodoini4, marca);
	fprintf(stream1," km=%d\n",D);
	fprintf(stream2,"%d\n",D);
	suma=suma+D;
	porcentaje(suma, num);
	suma=0;
	s=s+1;
	printf("sección de cruce\n");
	printf("de: ");
	while(1)
	{
		m1=random(num); //GENERACIÓN ALEATORIA DEL PRIMER PUNTO DE
				CRUCE
		m1++;
		m2=random(num); //GENERACIÓN ALEATORIA DEL SEGUNDO PUNTO DE
				CRUCE
		m2++;
		if(m1>m2)
		{
			mx=m1;
			m1=m2;
			m2=mx;
		}
		if(m1==m2)
			continue;
		break;
	}
	fprintf(stream1, " %d", m1);  //PRIMER PUNTO DE CRUCE
	fprintf(stream1, "-%d  ", m2);  //SEGUNDO PUNTO DE CRUCE

//APLICACIÓN DE LA TECNICA DE LA RULETA SESGADA
		al=aleatorio(); //PRIMER NÚMERO GENERADO ALEATORIAMENTE
		fprintf(stream1, "%.3f ", al);
		for(i=0; i<4; i++)
		{
			if(al < arr[i])
			{
				if(indica==0)
				{
				if(i==0)
					pad1=apnodoini4;
				if(i==1)
					pad1=apnodoini1;
				if(i==2)
					pad1=apnodoini2;
				if(i==3)
					pad1=apnodoini3;
				indica=1;
				al=aleatorio();//SEGUNDO NÚMERO GENERADO
						ALEATORIAMENTE
				fprintf(stream1, "%.3f", al);
				i=-1;
				}
				else
				{
				if(i==0)
					pad2=apnodoini4;
				if(i==1)
					pad2=apnodoini1;
				if(i==2)
```

```
                        pad2=apnodoini2;
                  if(i==3)
                        pad2=apnodoini3;
                  indica=0;
                  break;
                  }
            }
            else
                  continue;
      }
      for(i=0; i<num; i++)
      {
            ar1[i]=pad1->ciudad;
            ar2[i]=pad2->ciudad;
            if(pad2->apsig==NULL)
                  break;
            pad1=pad1->apsig;
            pad2=pad2->apsig;
      }
      while(pad2->apant != NULL)
      {
            pad1=pad1->apant;
            pad2=pad2->apant;
      }
   cruce1(m1, m2, pad1, pad2);  //PRIMER PAREJA SELECCIONADA A
CRUZAR
   marca=5;
   imprime(pad1, marca);
   recorrido(pad1, num);
   printf(" kms = %d\n",D);
   marca=6;
   imprime(pad2, marca);
   recorrido(pad2, num);
   printf(" kms = %d\n",D);

   for(i=0; i<num; i++)
      {
            ar3[i]=pad1->ciudad;
            ar4[i]=pad2->ciudad;

            pad1->ciudad=ar1[i];
            pad2->ciudad=ar2[i];
            if(pad2->apsig==NULL)
                  break;

            pad1=pad1->apsig;
            pad2=pad2->apsig;
      }
   while(pad2->apant != NULL)
      {
            pad1=pad1->apant;
            pad2=pad2->apant;
      }
      pad1=NULL;
      pad2=NULL;
   al=aleatorio(); //PRIMER NUMERO GENERADO ALEATORIAMENTE
   fprintf(stream1, "%.3f ", al);
   indica=0;
   for(i=0; i<4; i++)
      {
            if(al < arr[i])
```

```
                  {
                        if(indica==0)
                        {
                        if(i==0)
                              pad3=apnodoini4;
                        if(i==1)
                              pad3=apnodoini1;
                        if(i==2)
                              pad3=apnodoini2;
                        if(i==3)
                              pad3=apnodoini3;
                        indica=1;
                        al=aleatorio();//SEGUNDO NÚMERO GENERADO
ALEATORIAMENTE
                        fprintf(stream1, "%.3f\n", al);
                        i=-1;
                        }
                        else
                        {
                        if(i==0)
                              pad4=apnodoini4;
                        if(i==1)
                              pad4=apnodoini1;
                        if(i==2)
                              pad4=apnodoini2;
                        if(i==3)
                              pad4=apnodoini3;
                        indica=0;

                        break;
                        }
                  }
                  else
                        continue;
            }
      cruce1(m1, m2, pad3, pad4); //SEGUNDA PAREJA SELEECIONADA A
CRUZAR
      marca=7;
      imprime(pad3, marca);
      recorrido(pad3, num);
      printf(" kms = %d\n",D);
      marca=8;
      imprime(pad4, marca);
      recorrido(pad4, num);
      printf(" kms = %d\n",D);
      printf("\n");

      for(i=0; i<num; i++)
      {
            ar5[i]=pad3->ciudad;
            ar6[i]=pad4->ciudad;
            if(pad3->apsig==NULL)
                  break;
            pad3=pad3->apsig;
            pad4=pad4->apsig;
      }
      while(pad3->apant != NULL)
      {
            pad3=pad3->apant;
            pad4=pad4->apant;
      }
```

```
	pad3=NULL;
	pad4=NULL;
	pad1=apnodoini1;
	pad2=apnodoini2;

	for(i=0; i<num; i++)
		{
			pad1->ciudad=ar3[i];
			pad2->ciudad=ar4[i];
			if(pad2->apsig==NULL)
				break;
			pad1=pad1->apsig;
			pad2=pad2->apsig;
		}
		while(pad2->apant != NULL)
		{
			pad1=pad1->apant;
			pad2=pad2->apant;
		}
	pad3=apnodoini3;
	pad4=apnodoini4;
	for(i=0; i<num; i++)
		{
			pad3->ciudad=ar5[i];
			pad4->ciudad=ar6[i];

			if(pad3->apsig==NULL)
				break;
			pad3=pad3->apsig;
			pad4=pad4->apsig;
		}
		while(pad3->apant != NULL)
		{
			pad3=pad3->apant;
			pad4=pad4->apant;
		}
	apnodoini1=pad1;
	apnodoini2=pad2;
	apnodoini3=pad3;
	apnodoini4=pad4;
	aplica=1;

//PROCESO DE SELECCIÓN PARA ELEGIR AL INDIVIDUO QUE MUTARÁ
	if(s==3)
	{
		recorrido(apnodoini1, num);
		km1=D;
		recorrido(apnodoini4, num);
		km2=D;
		recorrido(apnodoini3, num);
		km3=D;
		recorrido(apnodoini2, num);
		km4=D;
		while(1)
		{
		if(km1>km2)
		{
			if(km1>km3)
			{
				if(km1>km4)
				{
```

```
                    if(km2>km3)
                    {
                          if(km2>km4)
                          {
                           mutacion(apnodoini1,apnodoini2);
                           break;
                          }
                    else{mutacion(apnodoini1,apnodoini4);
                               break;}
                    }
                    else if(km3>km4)
                    {
                     mutacion(apnodoini1,apnodoini3);
                     break;
                    }
                    else{
                         mutacion(apnodoini1,apnodoini4);
                         break;}
              }
              else if(km1>km2)
              {
                    if(km1>km3)
                    {
                     mutacion(apnodoini1,apnodoini4);
                     break;
                    }
                    else{
                         mutacion(apnodoini3,apnodoini4);
                         break;}
              }
              else if(km2>km3)
              {
               mutacion(apnodoini2,apnodoini4);
               break;
              }
                    else{
                         mutacion(apnodoini3,apnodoini4);
                         break;}
     }
}
  if(km2>km3)
{
     if(km2>km4)
     {
           if(km1>km3)
           {
                 if(km1>km4)
                 {
                  mutacion(apnodoini2,apnodoini1);
                  break;}
                  else{
                      mutacion(apnodoini2,apnodoini4);
                      break;}
           }
           else if(km3>km4)
           {
            mutacion(apnodoini2,apnodoini3);
            break;
           }
                 else{
                      mutacion(apnodoini2,apnodoini4);
```

```
                        break;}
      }
      else if(km1>km2)
            {
                  if(km1>km3)
                  {
                   mutacion(apnodoini4,apnodoini1);
                   break;
                  }
                  else{
                        mutacion(apnodoini4,apnodoini3);
                        break;}
             }
             else if(km2>km3)
             {
              mutacion(apnodoini4,apnodoini2);
              break;
             }
                  else{
                        mutacion(apnodoini4,apnodoini3);
                        break;}
}
if(km3>km4)
{
      if(km1>km2)
      {
            if(km1>km4)
            {
             mutacion(apnodoini3,apnodoini1);
             break;}
             else{
                 mutacion(apnodoini3,apnodoini4);
                 break;}
      }
      else if(km2>km4)
      {
       mutacion(apnodoini3,apnodoini2);
       break;
      }
            else{
                 mutacion(apnodoini3,apnodoini4);
                 break;}
}
else if(km1>km2)
      {
            if(km1>km3)
            {
             mutacion(apnodoini4,apnodoini1);
             break;}
             else{
                 mutacion(apnodoini4,apnodoini3);
                 break;}
      }
      else if(km2>km3)
      {
       mutacion(apnodoini4,apnodoini2);
       break;
      }
            else{
                   mutacion(apnodoini4,apnodoini3);
                 break;}
```

```
      }//while
      aplica=0;
      }//if
      }//for principal
      fclose(stream1);
      fclose(stream2);
      free(pad1);
      free(pad2);
      free(pad3);
      free(pad4);
      free(apnodoini1);
      free(apnodoini2);
      free(apnodoini3);
      free(apnodoini4);
}


void mutacion(TSP *m, TSP *mm)
{
TSP *apm=m, *apmm=mm;
int mut1, mut2, mut3, mut4, i=0;
mut1=apm->ciudad;
mut3=apmm->ciudad;
while(i<7)
{
      apm=apm->apsig;
      apmm=apmm->apsig;
      i++;
}
apmm=apmm->apsig;
mut2=apm->ciudad;
mut4=apmm->ciudad;
apm->ciudad=mut1;
apmm->ciudad=mut3;
apmm=apmm->apant;
i=0;
while(i<7)
{
      apm=apm->apant;
      apmm=apmm->apant;
      i++;
}
apm->ciudad=mut2;
apmm->ciudad=mut4;
while(apm->apant != NULL)
{
      apm=apm->apant;
      apmm=apmm->apant;
}
}


float aleatorio()
{
float al;
al=random(100);
al=al/100.0;
return(al);
}
```

```
void porcentaje(long suma, int num)
{
float km, div;
long sum;
sum=suma;
recorrido(apnodoini1, num);
km=D;
div=km/sum;
arr[0]=div;
recorrido(apnodoini2, num);
km=D;
div=km/sum;
arr[1]=div;
recorrido(apnodoini3, num);
km=D;
div=km/sum;
arr[2]=div;
recorrido(apnodoini4, num);
km=D;
div=km/sum;
arr[3]=div;
arr[1]=arr[0]+arr[1];
arr[2]=arr[1]+arr[2];
arr[3]=1.0;
fprintf(stream1, "%.3f, %.3f, %.3f, %.3f ", arr[0], arr[1], arr[2],
arr[3]);
}


int crear_km(int k, int i, int km, int flag)
{
   KM *apaux;
   if ((apaux = (KM *) malloc(sizeof(KM))) == NULL)
   {
     printf("Memoria insuficiente\n");
     exit(1);   }
   if (flag==1)
   {
     apaux->apsig=NULL;
     apaux->C1=k;
     apaux->C2=i;
     apaux->kms=km;
     apdistini=apaux;
     return(0);
   }
   if (apdistini->apsig == NULL)
   {
      apaux->apsig=NULL;
      apaux->C1=k;
      apaux->C2=i;
      apaux->kms=km;
      apdistini->apsig=apaux;
   }
   else
   {
      apdist=apdistini;
     while(apdist->apsig != NULL)
     {
        apdist=apdist->apsig;
     }
```

```
      apaux->apsig=NULL;
      apaux->C1=k;
      apaux->C2=i;
      apaux->kms=km;
      apdist->apsig=apaux;
   }
   return(0);
}


void imprime(TSP *prime, int marca)
{
TSP *ap=prime;
if(marca==1)
      fprintf(stream1,"Padre1 : ");
if (marca==2)
      fprintf(stream1,"Padre2 : ");
if(marca==3)
      fprintf(stream1,"Padre3 : ");
if(marca==4)
      fprintf(stream1,"Padre4 : ");
if(marca==5)
      printf("Hijo  1 : ");
if(marca==6)
      printf("Hijo  2 : ");
if(marca==7)
      printf("Hijo  1 : ");
if(marca==8)
      printf("Hijo  2 : ");
if(marca<5)
{
      while(ap != NULL)
      {
      fprintf(stream1, "%d ", ap->ciudad);
      ap=ap->apsig;
      }
}
else
{
while(ap != NULL)
      ap=ap->apsig;
}
printf("\n");
}


void cruce1(int m1, int m2, TSP *b1, TSP *b2)
{
      int aux, i, a=0, arr[10];
      TSP *apaux1=b1, *apaux2=b2, *ap1=apaux1;
      for(i=0; i<num; i++)
      {
            arr[i]=apaux1->ciudad;
            if(apaux1->apsig == NULL)
                  break;
            apaux1=apaux1->apsig;
      }

      while(apaux1->apant != NULL)
      {
      apaux1=apaux1->apant;
```

```
}
while(m1 != ap1->posicion)
      ap1=ap1->apsig;
while(apaux2->posicion != NULL)
{
      if(apaux2->ciudad != ap1->ciudad)
      {
            if(ap1->posicion < m2)
            {
                  ap1=ap1->apsig;
                  continue;
            }
            apaux1->ciudad=apaux2->ciudad;
            if(apaux1->apsig==NULL)
                  break;
            apaux1=apaux1->apsig;
      }
      while(ap1->posicion != m1)
      {
            ap1=ap1->apant;
      }
      if(apaux2->apsig==NULL)
                  break;
      apaux2=apaux2->apsig;
      while(apaux1->posicion >= m1 && apaux1->posicion <= m2)
      {
         if(apaux1->apsig==NULL)
         {
            a=1;
            break;
         }
         apaux1=apaux1->apsig;
      }
      if(a)
            break;
}
while(apaux2->apant != NULL)
{
      apaux2=apaux2->apant;
}
while(apaux1->apant != NULL)
{
      apaux1=apaux1->apant;
}
ap1=apaux2;
while(m1 != ap1->posicion)
      ap1=ap1->apsig;
i=0;
a=0;
while(arr[i] != NULL)
{
      if(arr[i] != ap1->ciudad)
      {
            if(ap1->posicion < m2)
            {
                  ap1=ap1->apsig;
                  continue;
            }
            apaux2->ciudad=arr[i];
            if(apaux2->apsig==NULL)
                  break;
```

```
                apaux2=apaux2->apsig;
            }

            while(ap1->posicion != m1)
            {
                ap1=ap1->apant;
            }
            if(arr[i+1]==NULL)
                        break;
            i++;
            while(apaux2->posicion >= m1 && apaux2->posicion <= m2)
            {
               if(apaux2->apsig==NULL)
               {
                  a=1;
                  break;
               }
               apaux2=apaux2->apsig;
            }
            if(a)
                  break;
      }

}


int crear_nodo(int i, int pos, int bandera, int flag)
{
   TSP *apaux;
   if ((apaux = (TSP *) malloc(sizeof(TSP))) == NULL)
   {
     printf("No hay memoria suficiente\n");
     exit(1);
   }
if(bandera==1)
{
   if (flag==1)
   {
     apaux->apsig=NULL;
     apaux->apant=NULL;
     apaux->ciudad=i;
     apaux->posicion=pos;
     apnodoini1=apaux;
     return(0);
   }
   if (apnodoini1->apsig == NULL)
   {
      apaux->apsig=NULL;
      apaux->ciudad=i;
      apnodoini1->apsig=apaux;
      apaux->posicion=pos;
      apaux->apant=apnodoini1;
   }
   else
   {
      apnodo1=apnodoini1;
     while(apnodo1->apsig != NULL)
     {
        apnodo1=apnodo1->apsig;
     }
      apaux->apsig=NULL;
```

```
      apaux->ciudad=i;
      apaux->posicion=pos;
      apnodo1->apsig=apaux;
      apaux->apant=apnodo1;
   }
}
if(bandera==2)
{
if (flag==1)
   {
     apaux->apsig=NULL;
     apaux->apant=NULL;
     apaux->ciudad=i;
     apaux->posicion=pos;
     apnodoini2=apaux;
     return(0);
   }
   if (apnodoini2->apsig == NULL)
   {
      apaux->apsig=NULL;
      apaux->ciudad=i;
      apaux->posicion=pos;
      apnodoini2->apsig=apaux;
      apaux->apant=apnodoini2;
   }
   else
   {
      apnodo2=apnodoini2;
     while(apnodo2->apsig != NULL)
     {
        apnodo2=apnodo2->apsig;
     }
      apaux->apsig=NULL;
      apaux->ciudad=i;
      apaux->posicion=pos;
      apnodo2->apsig=apaux;
      apaux->apant=apnodo2;
   }
}
if(bandera==3)
{
if (flag==1)
   {
     apaux->apsig=NULL;
     apaux->apant=NULL;
     apaux->ciudad=i;
     apaux->posicion=pos;
     apnodoini3=apaux;
     return(0);
   }
   if (apnodoini3->apsig == NULL)
   {
      apaux->apsig=NULL;
      apaux->ciudad=i;
      apaux->posicion=pos;
      apnodoini3->apsig=apaux;
      apaux->apant=apnodoini3;
   }
   else
   {
      apnodo3=apnodoini3;
```

```
     while(apnodo3->apsig != NULL)
     {
        apnodo3=apnodo3->apsig;
     }
      apaux->apsig=NULL;
      apaux->ciudad=i;
      apaux->posicion=pos;
      apnodo3->apsig=apaux;
      apaux->apant=apnodo3;
   }
}
if(bandera==4)
{
if (flag==1)
   {
     apaux->apsig=NULL;
     apaux->apant=NULL;
     apaux->ciudad=i;
     apaux->posicion=pos;
     apnodoini4=apaux;
     return(0);
   }
   if (apnodoini4->apsig == NULL)
   {
      apaux->apsig=NULL;
      apaux->ciudad=i;
      apaux->posicion=pos;
      apnodoini4->apsig=apaux;
      apaux->apant=apnodoini4;
   }
   else
   {
      apnodo4=apnodoini4;
     while(apnodo4->apsig != NULL)
     {
        apnodo4=apnodo4->apsig;
     }
      apaux->apsig=NULL;
      apaux->ciudad=i;
      apaux->posicion=pos;
      apnodo4->apsig=apaux;
      apaux->apant=apnodo4;
   }
}
   return(0);
}


void recorrido(TSP *prime, int num)
{
TSP *apaux=prime;
int origen, destino, dist, K=0, i, CI, CF, G1, GEN1, GEN2, defecto=0;
D=0;
CI=apaux->ciudad;
G1=apaux->posicion;
for(i=1; i<num; i++)
{
      origen=apaux->ciudad;
      apaux=apaux->apsig;
      destino=apaux->ciudad;
      K=distancia(origen, destino);
```

```
      if(K>defecto)
      {
            defecto=K;
            GEN1=(apaux->apant)->posicion;
            GEN2=apaux->posicion;
      }
D=K+D;
}
CF=apaux->ciudad;
K=distancia(CI, CF);
if(K>defecto)
      {
            defecto=K;
            GEN1=apaux->posicion;
            GEN2=G1;
      }
D=K+D;
if(aplica==1 && defecto>again)
{
      again=defecto;
      Genoma(prime, GEN1, GEN2, num);
}
again=0;
}


int distancia(int origen, int destino)
{
KM *apaux=apdistini;
int nume;
while(apaux != NULL)
{
if((origen==apaux->C1 || origen==apaux->C2) && (destino==apaux->C1 ||
destino==apaux->C2))
{
      nume=apaux->kms;
      return(nume);
}
else
{
      apaux=apaux->apsig;
      continue;
}//else
}//while
return(nume);
}


void Genoma(TSP *prime, int G1, int G2, int num)
{
TSP *apaux=prime;
int i, gm1, gm2;
for(i=0; i<num; i++)
{
      if(apaux->posicion==G1)
      {
            gm1=apaux->ciudad;
            if(apaux->apant==NULL)
            {
                  while(apaux->apsig != NULL)
                        apaux=apaux->apsig;
```

```
                  gm2=apaux->ciudad;
                  apaux->ciudad=gm1;
                  while(apaux->posicion != G1)
                        apaux=apaux->apant;
                  apaux->ciudad=gm2;
            }
            else{
            gm2=(apaux->apant)->ciudad;
            apaux->ciudad=gm2;
            (apaux->apant)->ciudad=gm1;
            }
      }
      if(apaux->posicion==G2)
      {
            gm1=apaux->ciudad;
            if(apaux->apsig==NULL)
            {
                  while(apaux->apant != NULL)
                        apaux=apaux->apant;
                  gm2=apaux->ciudad;
                  apaux->ciudad=gm1;
                  while(apaux->posicion != G2)
                        apaux=apaux->apsig;
                  apaux->ciudad=gm2;
            }
            else{
            gm2=(apaux->apsig)->ciudad;
            apaux->ciudad=gm2;
            (apaux->apsig)->ciudad=gm1;
            }
      }
      if(apaux->apsig!=NULL)
            apaux=apaux->apsig;
}
recorrido(prime, num);
}
```