UNIVERSIDAD NACIONAL DE LA PLATA

FACULTAD DE INFORMÁTICA

MAGISTER EN AUTOMATIZACIÓN DE OFICINAS

UNA SOLUCIÓN DE COMPUTACIÓN EVOLUTIVA PARA EL TSP, SU POSIBLE APLICACIÓN EN LAS ORGANIZACIONES

REALIZADA POR: GABRIELA F. MINETTI

DIRIGIDA POR: RAÚL GALLARD

PRESENTADA EN: OCTUBRE DE 2000

RESUMEN

Esta tesis investiga posibles incrementos de la performance en las soluciones de ciertos problemas de optimización combinatorial NP-duros. Ejemplos de esto son los problemas de secuenciamiento puros. Se realiza una reseña de los métodos convencionales, comparándolos con los pertenecientes al campo de la Computación Evolutiva, conjuntamente con la propuesta de eventuales mejoras a estos últimos. Las aplicaciones prácticas discutidas en esta tesis se encuentran fuertemente relacionadas con la administración, el diseño de redes en general, y el diseño de circuitos.

TABLA DE CONTENIDOS

LISTA DE FIGURAS

LISTA DE TABLAS

INTRODUCCIÓN

A diferencia de la filosofía y de la psicología, que también se ocupan de la inteligencia, la labor de la inteligencia artificial está orientada a la creación y entendimiento de entes inteligentes artificiales. La inteligencia artificial puede definirse como el esfuerzo para desarrollar sistemas basados en computadoras (software y hardware) que se comporte como un ente inteligente. Algunos sistemas inteligentes se apoyan en la experiencia y en el conocimiento humano, y en los patrones de razonamiento. Aunque las aplicaciones basadas en la inteligencia artificial son mucho más limitadas que la humana, son de gran interés por varias razones:

- Para preservar la experticia de las personas altamente capacitadas de una organización, en caso de que estas se distancien de la misma.
- Para almacenar información de forma activa -para crear una base de conocimiento organizacional- que los empleados pueden examinar o, aprender las reglas no escritas en los libros.
- Para crear un mecanismo que no esté sujeto a errores humanos causados, por ejemplo, por la fatiga, las preocupaciones, etc. Esto puede ser especialmente útil cuando las tareas se efectúan en ambientes peligrosos para las personas. Por la misma razón es provechoso en tiempos de crisis.
- Para eliminar tareas rutinarias e insatisfactorias para el hombre.
- Para sugerir soluciones a problemas específicos que son masivos y demasiado complejos para ser analizados por las personas en un corto período de tiempo, en función a la base de conocimiento de la organización.

En esta tesis se trabaja en una de las áreas pertenecientes a la inteligencia artificial: la computación evolutiva. La cual involucra una serie de técnicas para resolver una amplia variedad de problemas; basadas conceptualmente en el método que usan los organismos vivientes para adaptarse a su ambiente, el proceso de evolución. Algunas de las áreas en las que la computación evolutiva ofrece soluciones son las de: optimización, diseño de productos y monitoreo de sistemas industriales. Es en el área de optimización donde este trabajo se enfoca, en particular en el problema del viajante (TSP). Este es un problema de

secuenciamiento puro con una fuerte afinidad a los problemas de scheduling, en especial al de Flow-Shop Scheduling. El TSP es un interesante problema a resolver desde el punto de vista matemático y computacional. Además de utilizarse como herramienta en la administración de proyectos, diseño de redes, diseño de circuitos, entre otros.

El objetivo de esta tesis es lograr, mediante el estudio y el análisis de los mecanismos clásicos y de los evolutivos, un sustancial crecimiento en la performance de la solución a un problema de optimización combinatorial NP-duro -de amplia aplicación en la administración y en la industria-: el problema del viajante de comercio. Avalando esto con la presentación de una serie de resultados y su correspondiente análisis, obtenidos al incorporar posibles mejoras a un algoritmo evolutivo avanzado; diseñado especialmente, por Michalewicz, para resolver el TSP euclideano.

Esta tesis está divida en capítulos. El primero brinda una definición clara y concisa del problema del viajante así como una descripción de algunas de sus aplicaciones prácticas. En el segundo se introduce el concepto de computación evolutiva, conjuntamente con un detalle de los distintos tipos de sistemas que la conforman: Las Estrategias Evolutivas, La Programación Evolutiva, La Programación Genética, excepto Los Algoritmos Genéticos. Estos últimos son tratados en el capítulo 3 con mayor profundidad. En el cuarto se hace una reseña sobre las características de multiplicidad en algoritmos evolutivos avanzados. Mientras que en el capítulo 5 se describen una serie de métodos convencionales para resolver el TSP. La resolución del TSP mediante algoritmos genéticos se detalla en el sexto capítulo. Un algoritmo evolutivo avanzado diseñado especialmente para el TSP euclideano es descripto en el capítulo 7, se innova dicho algoritmo al aplicarle la característica de multiplicidad. Por último, se brindan las conclusiones generales de esta tesis.

CAPÍTULO 1: EL PROBLEMA DEL VIAJANTE DE COMERCIO

1.1. INTRODUCCIÓN

Aplicaciones, tales como: redes de comunicaciones, en especial redes de computadoras, diseño de rutas y redes en general, diseño de placas de circuitos impresos o PCB (Printed Circuit Board, en inglés), planificación y programación de actividades en proyectos de desarrollo de sistemas de cualquier tipo se analizan y llevan a cabo en áreas científicas e ingenieriles. Donde, resulta necesario establecer proyectos con el objetivo de realizar dichas aplicaciones. La disposición de un proyecto está dada, en gran medida, por:

- *La definición del proyecto*. Esto involucra las actividades de organización y formalización del proyecto, además de la definición de la estructura interna del mismo.
- *La planificación del proyecto*. En esta etapa se llevan a cabo las siguientes actividades: desarrollo de la estructura básica de trabajo, definición de la programación, estimación de los recursos, validación y redefinición de los objetivos del proyecto en cuestión, planificación de la administración de riesgos, y por último se realiza la aprobación o no de la planificación total. Siendo, los objetivos generales de la programación: definir de forma transparente la dependencia entre las tareas y el flujo de trabajo, además de determinar la duración total del proyecto. Esto se traduce en un problema de optimización al asignar distintas actividades a recursos, lo cual implica problemas de *scheduling*, propiamente dichos, o sus variantes tales como el caso del *problema del viajante*.
- *El seguimiento del proyecto*. Dicha etapa implica: el análisis de datos obtenidos durante el control de avance y las variaciones, la ejecución de acciones correctivas, la comunicación del estado, y por último el cierre del proyecto.

Las aplicaciones nombradas anteriormente corresponden a la puesta en práctica de problemas de optimización combinatorial NP-Completos. En especial, se identifican con el clásico problema del viajante de comercio; ya que tratan de

optimizar el costo incurrido al seguir una secuencia de nodos, *jobs*. Dicha secuencia forma un *circuito hamiltoniano*. A continuación se describe el problema del viajante de comercio, y luego se detallan las aplicaciones prácticas y su relación con este problema.

1.2. EL PROBLEMA DEL VIAJANTE DE COMERCIO

El problema del viajante de comercio o agente viajero, en inglés *Traveling Salesman Problem* (TSP), es uno de los problemas de optimización combinatorial NP-duros más ampliamente estudiado. Su declaración es engañosamente simple: un viajante busca el camino más corto para pasar por *m* ciudades. En otras palabras, una persona debe visitar un conjunto de *m* ciudades, comenzando en una ciudad determinada y finalizando en la misma ciudad; luego de haber visitado todas ellas sólo una vez. Esto significa que nunca regresa a una ciudad ya visitada, excepto la primera.

Este problema puede ser representado por un grafo, cuyos nodos representan cada ciudad y los arcos la distancia entre un par de ellas; formando, de esta manera, un ciclo hamiltoniano.

El objetivo es encontrar la secuencia de visitas óptima; la cual puede ser evaluada según distintos criterios, como por ejemplo: la minimización del costo o del tiempo, la maximización de la velocidad.

El TSP puede ser simétrico o asimétrico. En el primer caso, dado un conjunto de *m* nodos y distancias para cada par de nodos, la distancia desde el nodo *i* al nodo *j* es la misma que la distancia desde el nodo *j* al nodo *i*. Mientras que en el caso del TSP asimétrico, la distancia desde el nodo *i* al nodo *j* es distinta a la distancia desde el nodo *j* al nodo *i* [116].

Existen varias formulaciones matemáticas del problema. La presentada a continuación usa relativamente pocas variables, y define las variables cero-uno de la siguiente manera:

$$x_{ij} = \begin{cases} 1 \text{ si la ruta incluye la secuencia de ir desde } i \text{ a } j, \\ 0 \text{ en otro caso.} \end{cases}$$

para todo *i* y *j*.

El objetivo es minimizar,

$$\sum_{i=1}^{m}\sum_{j=1}^{m} c_{ij}\, x_{ij} \quad \text{donde } c_{ii} = \infty \text{ para } i = 1,....,m$$

sujeto a:

$$\sum_{i=1}^{m} x_{ij} = 1 \qquad \text{para } i = 1,....,m \qquad (1)$$

$$\sum_{j=1}^{m} x_{ij} = 1 \qquad \text{para } j = 1,....,m \qquad (2)$$

$$x_{ij} \in \ Z^{+} \ \forall \ i \text{ y } j \qquad (3)$$

Las restricciones (1), (2) y (3) aseguran que cada x_{ij} sea cero o uno. La restricción (1) garantiza un arribo a cada ciudad, mientras que la (2) requiere que un *tour* incluya una salida desde cada ciudad.

1.3. ALGUNAS APLICACIONES PRÁCTICAS DEL TSP

1.3.1. PROBLEMAS DE SCHEDULING

Los problemas de scheduling son, en reiteradas ocasionas, muy complicados de resolver por contar con un gran número de: actividades relacionadas y restringidas a cada una de las otras, recursos a actividades, y los recursos o actividades a eventos externos del sistema [80]. La función de evaluación matemática, para este problema, es altamente compleja y el espacio de búsqueda es muy amplio con un gran número de óptimos locales. La similitud entre el TSP y el problema de secuenciamiento radica en que los jobs pueden verse como ciudades y los tiempos de *set-up* (configuraciones) dependientes de la secuencia pueden apreciarse como la distancia entre dichas ciudades [97]. El objetivo es determinar una secuencia de jobs (un schedule), de forma tal que todos ellos se lleven a cabo en una cantidad mínima de tiempo. Es necesario destacar, que el costo de pasar del job A

al B, puede ser diferente al costo de recorrer el camino inverso, esto es desde el job B al A. Consecuentemente, es necesario resolver este problema por medio de la variante asimétrica del TSP.

El problema del secuenciacimiento de jobs con set-up dependiente de la secuencia se presenta en distintas áreas, como por ejemplo: en la programación de un proyecto de desarrollo de sistemas. También es posible hallar un caso más específico en la televisión, cuando es necesario planificar un secuenciaciamiento óptimo de los comerciales durante un corte publicitario [49]. Otro campo es el que comprende el diseño de PCB, para más detalles ver el punto 1.2.5.

Otra aplicación se encuentra en el problema de secuenciamiento de aviones, en inglés *Aircraft Sequencig Problem* (ASP) [94]. Se asume que hay *n* aviones esperando el permiso correspondiente para aterrizar en un única pista. Las regulaciones de seguridad requieren de un cierto tiempo entre dos aterrizajes. Este tiempo depende de los aviones, clasificados por tipo. El objetivo es encontrar una secuencia de aterrizaje, de forma tal que se logre minimizar la cantidad total de tiempo.

1.3.2. REDES Y TELECOMUNICACIONES

En *routing*, uno de los problemas más conocidos es el del viajante o TSP, en el cual es necesario optimizar la ruta a recorrer fijando algún criterio para ello. En función de lo anterior se torna prioritario encontrar un adecuado balance entre la exactitud de los resultados y el tiempo consumido para obtenerlos, es decir entre eficiencia y eficacia.

El TSP encuentra su aplicación más directa en el área de las empresas de telecomunicaciones; ya que estas necesitan optimizar las rutas de las redes telefónicas para minimizar costos.

Otro caso de uso del problema del viajante de comercio altamente relacionado con el área de comunicaciones, es el diseño de una red de fibra óptica [24]. En la cual conexiones punto a punto son necesarias, siendo una de las topologías posibles para este tipo de red: la de anillo. En este caso el objetivo es minimizar la distancia recorrida por la unión de todos los nodos involucrados y así abaratar

los costos de tendido de la red. Donde la topología de anillo implica un conjunto de enlaces punto a punto [138]; por los cuales circula un patrón de bits especial, denominado *ficha* o *token*, cuando todas las estaciones están inactivas. En el momento en que algún nodo necesite transmitir toma la ficha y la retira del anillo antes de emitir. El mensaje enviado debe pasar por cada uno de los nodos sólo una vez y regresar al lugar de origen. Una vez que una estación terminó de transmitir, esta debe regenerar la ficha.

El TSP puede, también, adaptarse para encontrar el *path* o ruta más corta entre dos nodos en una red de computadoras, conociendo por anticipado los costos de enlaces y conectividad existentes.

1.3.3. RED DE RECOLECCIÓN DE RESIDUOS

Una de las herramientas usadas en un proyecto ejecutivo de manejo y disposición de residuos sólidos municipales es el algoritmo para resolver el problema del viajante de comercio [129]. Este proyecto debe cumplimentar un conjunto de etapas:

- Diagnóstico de la situación actual de la localidad con respecto al manejo y disposición de los residuos sólidos.
- Evaluación de parámetros.
- Diseño del Sistema de Manejo de los Residuos Sólidos Municipales.

Es en la última etapa donde resulta necesario, entre otras cosas, diseñar el transporte y recolección de los residuos teniendo en cuenta los siguientes factores: economía del sistema, resistencia al cambio por parte de los usuarios del sistema y/o de los prestadores del servicio, topografía de la localidad, trazo y vialidad establecidas en la localidad. Resultando posibles los siguientes métodos de recolección: de llevar y traer, de parada fija y de contenedores. En los dos últimos el modelo a aplicar para optimizar el proceso de recolección, es el algoritmo para resolver el TSP.

1.3.4. FILM-COPY DELIVERER PROBLEM (FDP)

El *film-copy deliverer problem* es un nuevo caso de optimización combinatorial examinado por Cheng y Gen [20]. El FDP se formula a continuación. La copia de un film se requiere para proyectarse en *n* cines. El número de veces que cada cine presenta el film se predetermina y denota por un entero no negativo $d_i (i = 1, 2, \ldots\ldots, n)$. Cada par de cines se conecta a través de un arco de longitud w_{ij}; y si los cines *i* y *j* no están directamente comunicados, entonces $w_{ij} = +\infty$.

El problema radica en encontrar un tour para el repartidor (deliverer), comenzando y finalizando el recorrido en el cine de partida, así la longitud total del tour se minimiza bajo la restricción que el repartidor debe entregar la copia al cine *i-ésimo* exactamente pero no consecutivamente d_i veces $(i = 1, 2, \ldots\ldots, n)$. Cuando todos los d_i son restringidos a 1, el FDP se convierte en un TSP, pero también puede ser generalizado a una amplia variedad de problemas de ruteo y de scheduling. Zhang y Zheng en [151] investigan la transformación del FDP al TSP y resuelven el problema transformado con las heurísticas conocidas para el TSP.

El FDP puede ser fácilmente representado con un grafo $G = (V, E)$, donde $V = \{v_1, v_2, \ldots\ldots, v_n\}$ es un conjunto finito de vértices representando a cada cine y $E = \{e_{ij} \mid e_{jj} = (v_i, v_j), v_i, v_j \in V\}$ es un conjunto finito de arcos representando la conexión entre dos cines. Cada arco tiene asociado un valor real no negativo, denotado por $W = \{w_{ij} \mid w_{ij} = w(v_i, v_j), w_{ij} > 0, v_i, v_j \in V\}$ representando las distancias. Cada vértice está asociado a un entero positivo que indica el número de veces que se presenta en cada cine y se denota por $D = \{d i \mid d_i > 0, d_i \in Z, i = 1, 2, \ldots\ldots, n\}$.

1.3.5. PLACA DE CIRCUITOS IMPRESOS (PCB)

El TSP también puede aplicarse en el diseño de un PCB. Esto es, asumiendo que un brazo de robot introduce *n* partes en puntos específicos de un PCB. Cada una de las partes pertenece a un cierto tipo *K* de partes. En un receptáculo se almacenan todas las de un determinado tipo. Los *K* recipientes se ubican a lo largo de uno de los laterales del PCB. Además, se supone que el brazo del robot

sólo puede hacer movimientos secuenciales en dirección horizontal o vertical, entonces el tiempo de viaje del brazo del robot de una ubicación a otra es proporcional a la distancia lineal entre los dos puntos. El problema del tour del robot (en inglés Robot Tour Problem, RTP) es encontrar una secuencia óptima de inserciones de las partes en el PCB, dadas las posiciones de los *n* puntos de inserción y las ubicaciones de los *K* receptáculos [18, 78].

Para ubicaciones fijas de los *K* recipientes, es posible transformar el RTP en un TSP para los *n* puntos de inserción. Obviamente, la distancia desde un punto de inserción a otro es igual a la distancia desde el primer punto de inserción al recipiente que contiene la parte a ser incorporada en el segundo punto más la distancia desde este recipiente al segundo punto de inserción. Es posible observar que la segunda parte de esta distancia es fija, esto significa que, es independiente de la secuencia de la distancia recorrida para llegar a ese receptáculo. Sin embargo la primer parte es dependiente de la secuencia. Esto es, todos los recipientes se encuentran ubicados en el mismo lado del PCB y las distancias son medidas en forma rectilínea, entonces la distancia dependiente de la secuencia es la que va desde la proyección del punto de inserción en ese lateral del PCB hasta el próximo recipiente.

CAPÍTULO 2: COMPUTACIÓN EVOLUTIVA

2. INTRODUCCIÓN

El principio de la evolución [58] se origina en el concepto básico de la biología, donde cada criatura en la cadena es el producto de una serie de "accidentes" que se han elegido bajo la presión selectiva que ejerce el medio ambiente. Una gran cantidad de generaciones, la variación aleatoria y la selección natural muestran el comportamiento de individuos y especies para ajustarse a las demandas de sus vecinos.

Este ajuste puede ser bastante extraordinario y determinante, una clara indicación de esto es la creatividad de la evolución. Mientras que la evolución no tiene un propósito intrínseco (ésta es, meramente el efecto de la actuación de las leyes físicas sobre y con las poblaciones y especies), sino que es capaz de dar soluciones ingenieriles a los problemas de sobrevivencia que son únicas para las circunstancias de cada individuo y bastante ingeniosas.

El proceso evolutivo dentro de una computadora provee un medio para solucionar problemas ingenieriles complejos (involucrando disturbios caóticos, aleatoriedad y dinámicas no lineales complejas) que los algoritmos tradicionales no han sido capaces de conquistar [7, 8, 108, 128]. Por esta razón, el campo de la computación evolutiva es una de las áreas de más rápido crecimiento de la ciencia de la computación y de la ingeniería. Es decir, apunta a muchos problemas que fueran previamente alcanzados, tales como un diseño rápido de medicinas, y el análisis veloz de tácticas de combate para defensa. Potencialmente, este campo puede completar el sueño de la inteligencia artificial: una computadora que pueda aprender y se convierta en un experto en cualquier área elegida.

En términos más generales, la evolución puede describirse como un proceso iterativo de dos pasos, consistentes de una variación aleatoria seguida de una selección. El enlace entre esta descripción de la evolución y los algoritmos de optimización que son la estrella de la computación evolutiva es conceptualmente simple.

Al igual que la evolución natural que comienza desde una población inicial de criaturas, la opción algorítmica comienza al seleccionar un conjunto inicial de

soluciones para un problema en particular. El conjunto puede ser elegido al generar soluciones aleatorias o al utilizar cualquier conocimiento disponible sobre el problema.

Estas soluciones denominadas "*padres*" entonces generan "*hijos*" por medio de variaciones aleatorias. Estas soluciones resultantes se evalúan en función de su efectividad o aptitud (su *fitness*) y la selección a las que son sometidas. Tal como en la naturaleza se impone la regla "sobrevive el más fuerte", entonces las soluciones de menor fitness se eliminan, y el proceso es repetido sobre las sucesivas generaciones.

Los algoritmos tradicionales usados para descubrir las soluciones más apropiadas (algoritmos de optimización) requieren, para ser útiles, que sus usuarios realicen muchas asumpciones de cómo evaluar el fitness de una solución. A estos medios de evaluación tradicionales se los puede denominar: fitness, función de costo, superficie de respuesta, o, en ingeniería índice de performance. Por ejemplo, los algoritmos de programación lineal demandan que las funciones de costo también sean lineales. Otra opción tradicional, es la búsqueda basada en el gradiente; en el cual el punto de gradiente cero indica que el máximo o el mínimo es alcanzado. Esta última opción requiere una función de costo diferenciable.

Pero los algoritmos evolutivos no requieren tales asumpciones. Fundamentalmente, el índice de *performance* necesita ser capaz de ordenar de acuerdo a un rango a dos soluciones competidoras, esto significa que debe determinar cuál solución es la mejor de ambas. Esto hace que una amplia variedad de problemas que están fuera del rango de la ingeniería convencional sean atacados por la opción evolutiva. Concluyendo, los algoritmos evolutivos pueden a menudo resolver problemas que las técnicas numéricas no logran solucionar.

La alternativa evolutiva ofrece grandes ventajas. Una de ellas es la capacidad para adaptarse a situaciones cambiantes. Por ejemplo, el caso en que un administrador deba encontrar la mejor planificación para la operación de una fábrica. Sin dudas, habrá muchas restricciones: disponibilidad de personal, el número de máquinas, el tiempo requerido para configurar los cambios en las máquinas, entre otras. Aún si el administrador encontrase la planificación óptima,

sería deseable contar con planificaciones alternativas que tuviesen en cuenta, por ejemplo, la rotura de máquinas, la disminución del personal, etc.; para poder obtener el o los productos a tiempo. En el mundo real, el problema puede ser significativamente divergente del planteado originalmente.

Desafortunadamente, en muchos de los procedimientos de optimización tradicionales, el cálculo debe ser iniciado nuevamente desde el comienzo si una de las variantes del problema cambia. Esto es computacionalmente muy caro. Con un algoritmo evolutivo la población actual actúa como una reserva de conocimiento almacenado que puede ser aplicado a ambientes dinámicos. No siendo necesario iniciar el cálculo desde ningún punto.

Otra ventaja que presenta la opción evolutiva es la siguiente: puede generar soluciones lo suficientemente buenas rápidamente para su uso. Esta habilidad es bien ilustrada en el problema del viajante. Este problema pertenece a la familia de los problemas NP-duros, donde NP hace referencia a los problemas donde no puede hallarse un algoritmo que lo resuelva en tiempo polinomial. Por lo que puede ser imposible encontrar el óptimo global en algunas aplicaciones, o no haber una manera de chequear si una solución es un óptimo global. En muchos casos encontrar el óptimo global puede ser innecesario. Pero hallar una buena solución (un óptimo local) velozmente puede ser más deseable que encontrar la mejor solución (el óptimo global) lentamente.

Desde hace aproximadamente treinta años [107] existe un creciente interés en los sistemas de resolución de problemas basados en los principios de evolución y hereditarios, tales como: mantener una población de soluciones potenciales (que como se menciona anteriormente sufren algún proceso de selección basado en el fitness de individuos), y algunos operadores “genéticos”. Una clase de tales sistemas son las *estrategias evolutivas*. Estos algoritmos imitan los principios de la evolución natural para problemas de optimización de parámetros [114, 126] (Rechenberg, Schwefel). Otro tipo es la *programación evolutiva* [51] que introduce Fogel; es una técnica para buscar a través de un espacio pequeño de máquinas de estados finitos. Las técnicas de búsqueda *Scatter* que han sido presentadas por Glover [64] mantienen una población de puntos de referencia y genera vástagos por combinaciones lineales de peso. Otro tipo de sistemas

basados en la evolución son los *algoritmos genéticos* introducidos por Holland 1975 [82]. En 1990, Koza [95] propone la *programación genética*, mediante la cual se busca el mejor programa computacional para resolver un problema en particular.

En este capítulo se describen las estrategias evolutivas, la programación evolutiva y la programación genética en las secciones 2.3, 2.4 y 2.5 respectivamente. Mientras que, los algoritmos genéticos son tratados con mayor profundidad en el capítulo 3.

3. DESCRIPCIÓN DE UN ALGORITMO EVOLUTIVO GENÉRICO

Se usa el término Algoritmo Evolutivo, para todos los sistemas basados en la evolución (incluyendo los sistemas descriptos en la sección anterior).

El algoritmo evolutivo es probabilístico y mantiene una población de individuos, $P(t) = \{x^t_1, \ldots\ldots\ldots\ldots, x^t_n\}$ para la iteración t [107]. Cada individuo (o cromosoma) representa una solución potencial al problema en cuestión, y, un algoritmo evolutivo es implementado como alguna estructura de datos S. Cada solución x^t_i es evaluada para dar alguna medida de su "fitness". Entonces, una nueva población (iteración t+1) es formada al seleccionar los individuos de mejor fitness (seleccionar). Algunos miembros de la nueva población sufren transformaciones por medio de operadores "genéticos" (alterar) para construir nuevas soluciones. Existen transformaciones unarias m_i (mutación), las cuales crean nuevos individuos al realizar pequeños cambios en un único individuo ($m_i : S \rightarrow S$), y transformaciones de orden más alto c_j (*crossover*), el cual crea nuevos individuos al combinar partes de (dos o más) soluciones distintas ($c_j : S \times \ldots \times S \rightarrow S$). Después de algunas generaciones el programa converge, siendo esperado que el mejor individuo represente una solución razonablemente cercana al óptimo. La figura 2.1 muestra el esquema correspondiente al algoritmo evolutivo.

Distintos sistemas evolutivos surgen a partir de esta idea del algoritmo evolutivo. Las principales diferencias entre ellos están ocultas en un nivel inferior: el uso de una estructura de datos apropiada (para la representación del cromosoma) junto con un conjunto expandido de operadores genéticos. Por

ejemplo los algoritmos genéticos clásicos usan *strings* binarios de longitud fija como un cromosoma (estructura de datos *S*) para sus individuos, y dos operadores genéticos como la mutación y el crossover binarios. En otras palabras, la estructura de un algoritmo genético es la misma que la estructura de un algoritmo evolutivo (figura 2.1) estando ocultas las diferencias en niveles inferiores. En la Programación Evolutiva los cromosomas no necesitan ser representadas por strings de bits y el proceso de alteración incluye otros operadores genéticos apropiados para la estructura y el problema dado.

```
procedimiento algoritmo evolutivo
begin
    t ← 0
    inicializar P ( t )
    evaluar P ( t )
    while (no se cumpla la condición de terminación) do
    begin
        t ← t + 1
        seleccionar P ( t ) desde P ( t – 1 )
        alterar P ( t )
        evaluar P ( t )
    end
end
```

Figura 2.1. La Estructura de un algoritmo evolutivo

Una representación natural de una solución potencial para un problema dado más una familia de operadores genéticos aplicables pueden ser bastantes útiles en la aproximación de soluciones de muchos problemas, y esta opción evolutiva es una dirección prometedora para resolver problemas en general.

4. ESTRATEGIAS EVOLUTIVAS

Las estrategias evolutivas han sido desarrolladas como un método para resolver problemas de optimización de parámetros [25, 128]; consecuentemente, un cromosoma representa a un individuo como un par de vectores,

$$\mathbf{v} = (\mathbf{x}, \boldsymbol{\sigma}).$$

Las primeras estrategias evolutivas se han basado en una población consistente de un único individuo y en un solo operador genético, la mutación. Sin embargo

la idea interesante (ausente en los algoritmos genéticos) es la representación de un individuo como un par de vectores reales, $\mathbf{v}=(\mathbf{x},\boldsymbol{\sigma})$. Donde **x** representa un punto en el espacio de búsqueda, y el segundo vector $\boldsymbol{\sigma}$ es un vector de desviaciones estándares. Las mutaciones se realizan al reemplazar **x** por:

$$\mathbf{x}^{t+1}=\mathbf{x}^{t}+N(0,\boldsymbol{\sigma}),$$

donde $N(0,\boldsymbol{\sigma})$ es un vector de números Gausianos independientes con una media igual a cero y una desviación estándar igual a $\boldsymbol{\sigma}$. (Esto concuerda con la observación biológica de que los cambios pequeños ocurren más a menudo que los grandes). El vástago (o individuo mutado) se acepta como un nuevo miembro de la población (el cual reemplaza a su padre) sí y solo sí tiene el fitness más alto y satisface todas las restricciones. De lo contrario, se elimina al hijo y la población se mantiene sin cambios. El vector estándar de desviaciones $\boldsymbol{\sigma}$ permanece sin modificaciones durante el proceso de evolución.

La idea principal detrás de estas estrategias es permitir el control de los parámetros, tal como, la varianza de la mutación; para así auto-adaptarse en lugar de modificar sus valores por algún algoritmo detérminístico [127].

Las estrategias evolutivas de mayor desarrollo son las que introduce Schwefel [124, 125, 126]:

1. la estrategia evolutiva $(\mu+\lambda)$ y,
2. la estrategia evolutiva (μ,λ).

En la estrategia $(\mu+\lambda)$, μ individuos producen λ hijos. La nueva población de $(\mu+\lambda)$ individuos es reducida por un proceso de selección de μ individuos. Mientras que la estrategia (μ,λ), los μ individuos producen λ hijos con $\lambda>\mu$, y el proceso de selección elige una nueva población de μ individuos del conjunto de λ hijos solamente. Consecuentemente, la vida de cada individuo es limitada a una generación. Esto permite que la estrategia evolutiva (μ,λ) actúe mejor en problemas con un óptimo cambiante en el tiempo, o en problemas donde la función es ruidosa (noisy).

Los operadores usados en estas dos estrategias incorporan aprendizaje en dos niveles: el parámetro de control $\boldsymbol{\sigma}$ no permanece constante, ni se modifica de

forma determinística, pero se incorpora en la estructura de individuos y sufre el proceso de evolución. Para producir un hijo, el sistema actúa en varias etapas:

1º. Selecciona dos individuos,

$(\mathbf{x}^1,\boldsymbol{\sigma}^1) = ((\mathbf{x}_1^1,.....,\mathbf{x}_n^1), (\boldsymbol{\sigma}_1^1,.....,\boldsymbol{\sigma}_n^1))$ y

$(\mathbf{x}^2, \boldsymbol{\sigma}^2)= ((\mathbf{x}_1^2,.....,\mathbf{x}_n^2), (\boldsymbol{\sigma}_1^2,.....,\boldsymbol{\sigma}_n^1))$.

Luego si $\mu > 1$ (es decir, si el tamaño de la población es mayor a uno), aplica un operador de recombinación (crossover) a los dos vectores. El crossover sobre estos grupos de información procede independientemente de cada uno de los otros.

Una variedad de mecanismos de recombinación son actualmente usados en las estrategias evolutivas y los operadores son sexuales y panmícticos. En los primeros, se recombinan dos individuos elegidos al azar desde la población de padres, donde se permite seleccionar dos veces al mismo individuo para la creación de un hijo. En cambio, la opción panmíctica de recombinación un padre elegido aleatoriamente se mantiene fijo mientras que para cada componente de sus vectores el segundo padre se selecciona al azar desde la población completa. En otras palabras, la creación de un solo hijo puede involucrar a todos los individuos padres.

Los tipos de crossovers tradicionales para las estrategias evolutivas son:

- el crossover discreto, donde el nuevo hijo es:

  $(\mathbf{x}, \boldsymbol{\sigma})= ((\mathbf{x}_1^{q_1},.....,\mathbf{x}_n^{q_n}),(\boldsymbol{\sigma}_1^{q_1},.....,\boldsymbol{\sigma}_n^{q_n}))$,

  y $\mathbf{q}_i = 1$ ó $\mathbf{q}_i = 2$ (cada componente proviene desde el primer o segundo padre preseleccionado),

- el crossover intermedio, donde el nuevo hijo es:

$(\mathbf{x}, \boldsymbol{\sigma})= (((\mathbf{x}_1^1 + \mathbf{x}_1^2)/2,....,(\mathbf{x}_n^1 + \mathbf{x}_n^2)/2),((\boldsymbol{\sigma}_1^1 + \boldsymbol{\sigma}_1^2)/2,....,(\boldsymbol{\sigma}_n^1 + \boldsymbol{\sigma}_n^2)/2))$.

Cada uno de estos operadores puede también aplicarse, en un modo global, donde el nuevo par de padres se seleccionan para *cada* componente del vector hijo.

Los operadores tradicionales pueden implementarse bajo la forma sexual o bajo la forma panmíctica.

2º. Aplica la mutación al hijo $(\mathbf{x}, \boldsymbol{\sigma})$ obtenido y el nuevo individuo resultante es $(\mathbf{x'}, \boldsymbol{\sigma'})$, donde:

$$\boldsymbol{\sigma'} = \boldsymbol{\sigma} \cdot e^{N(0,\Delta_\sigma)}$$

$$\mathbf{x'} = \mathbf{x} + N(0, \boldsymbol{\sigma'}),$$

siendo Δ_σ un parámetro del método.

Este mecanismo de mutación involucra a los propios parámetros de la estrategia evolutiva (desviación estándar y covarianzas) durante la búsqueda, explotando un enlace implícito entre un modelo interno apropiado y buenos valores de fitness. La evolución y la adaptación alcanzada de los parámetros de la estrategia de acuerdo a los requerimientos topológicos la define Schwefel como *auto-adaptación* [127].

5. PROGRAMACIÓN EVOLUTIVA

La técnica original de Programación evolutiva ha sido desarrollada por Lawrence Fogel [57]. Apunta a la evolución de la inteligencia artificial en el sentido de desarrollar la habilidad de predecir cambios en un medio ambiente. Dicho contexto ha sido descripto como una secuencia de símbolos (desde un alfabeto finito) y un supuesto algoritmo evolutivo que produce un nuevo símbolo, como salida. La salida debería maximizar la función de retribución (payoff), la cual mide la exactitud de la predicción. Por ejemplo, se puede considerar una serie de eventos, marcados por los símbolos $a_1, a_2, ...$; un algoritmo debería predecir el próximo símbolo (desconocido), $a_1, a_2, ..., a_n$. La idea de la programación evolutiva es evolucionar tales algoritmos.

Las máquinas de estados finitos han sido seleccionadas como una representación cromosomática de individuos; porque, estas máquinas proveen una representación significativa del comportamiento basado en la interpretación de símbolos. La figura 2.2 brinda un ejemplo de un diagrama de transición de una única máquina de estados finitos para un chequeo de paridad. Tal diagrama de transición presenta grafos dirigidos que contienen un nodo por cada estado, arcos que indican la transición desde un estado a otro, y valores de entrada y salida.

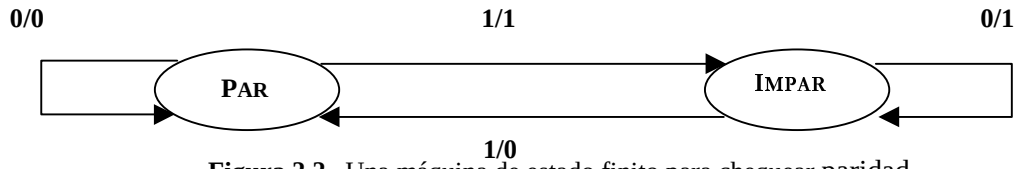

Figura 2.2. Una máquina de estado finito para chequear paridad

Aquí existen dos estados Par e Impar (la máquina inicia en el estado Par); la máquina reconoce una paridad de un string binario.

Entonces, la técnica de la programación evolutiva mantiene una población de máquinas de estados finitos, donde cada individuo representa una solución potencial al problema (por ej. representa un comportamiento en particular). Cada máquina de estado finito es evaluada para dar alguna medida de su fitness. Esto es hecho de la siguiente manera: cada máquina es expuesta al medio ambiente ya que examina todos los símbolos vistos previamente. Para cada secuencia $a_1, a_2, \ldots\ldots, a_i$ produce una salida a'_{i+1} la cual se compara con el próximo símbolo observado, a_{i+1}.

Al igual que las estrategias evolutivas, esta técnica primero crea un hijo y más tarde selecciona los individuos para la próxima generación. Cada padre produce un solo hijo. Los vástagos son creados a través de la mutación aleatoria de la población de padres (ver figura 2.3). Los operadores de mutación posibles son cinco:

1. Cambio de un símbolo de salida.
2. Cambio de una transición de estados.
3. Adición de un estado.
4. Eliminación de un estado.
5. Cambio del estado inicial.

Estas mutaciones son elegidas con respecto a alguna distribución de probabilidad (las cuales pueden cambiar durante el proceso de evolución); también es posible aplicar más de una mutación a un único padre (se toma una decisión sobre el número de mutaciones con respecto a alguna otra distribución de probabilidad).

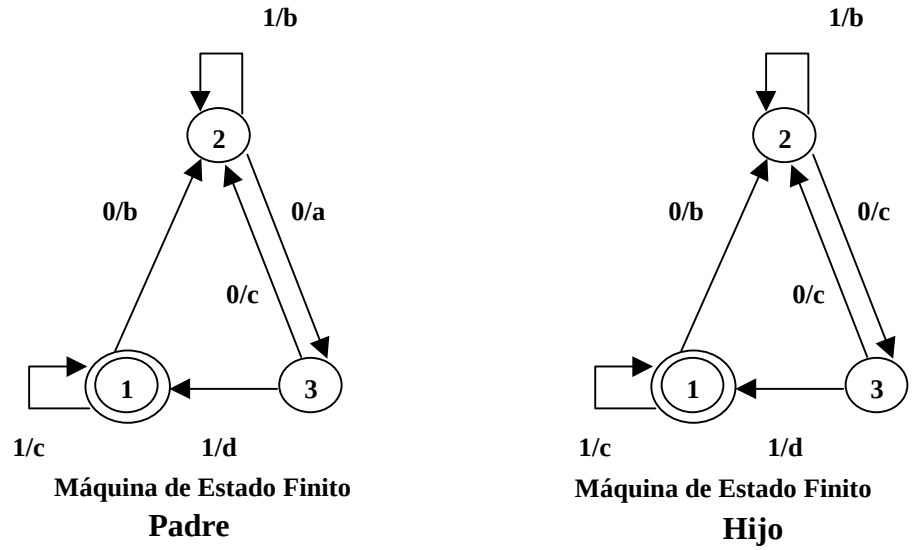

Figura 2.3. Una máquina de estado finito y su hijo. Las máquinas inician en el estado 1.

Para la próxima generación se retienen los mejores μ individuos; para poder pertenecer a la próxima generación un individuo debería estar en la cima del 50% de la población intermedia. En la versión original [57] este proceso se itera varias veces antes de que el próximo símbolo de salida esté disponible. Una vez que un nuevo símbolo resulta accesible, se agrega a la lista de símbolos conocidos, y se repite el proceso entero.

Recientemente las técnicas de la programación evolutiva han sido generalizadas para manejar problemas de optimización numéricos; para más detalles ver [52, 55].

6. PROGRAMACIÓN GENÉTICA

Koza [94, 95] recientemente ha sugerido otra interesante alternativa. Koza propone que el programa deseado debería evolucionarse a sí mismo durante el proceso de evolución. En otras palabras, en vez de resolver un problema y construir un programa evolutivo para solucionarlo, sería necesario buscar el espacio de programas computacionales posibles para resolver el problema de la mejor manera. Koza ha desarrollado una nueva tecnología, denominada Programación genética, que provee una manera de llevar a cabo tal búsqueda.

Los cinco principales pasos en el uso de la programación genética para un problema particular son:

1. Selección de terminales.
2. Selección de una función.
3. Identificación de la función de evaluación.
4. Selección de parámetros del sistema.
5. Selección de la condición de terminación.

Es importante observar que la estructura que sufre la evolución es un programa computacional estructurado jerárquicamente. El espacio de búsqueda es un hiperespacio de programas válidos, que pueden ser vistos como un espacio de árboles. Cada uno de ellos está compuesto de funciones y terminales apropiadas para el dominio de un problema en particular; el conjunto de todas las funciones y terminales se selecciona *a priori* de tal manera que alguno de estos árboles produzca una solución.

Por ejemplo, dos estructuras e_1 y e_2 (figura 2.4) representan expresiones $2x + 2.11$ y $x.\text{sen}(3.28)$, respectivamente. Un posible hijo e_3 (después del crossover entre e_1 y e_2) representa $x.\text{sen}(2x)$.

La población inicial está compuesta de tales árboles; la construcción de un árbol es sencilla. La función de evaluación asigna un valor de fitness que evalúa la performance de un árbol (programa). La evaluación se basa en un conjunto preseleccionado de casos de prueba. En general, la función de evaluación retorna la suma de distancias entre los resultados correctos y los obtenidos sobre todos los casos de prueba. La selección es proporcional; cada árbol tiene una probabilidad proporcional a su fitness de ser elegido para la próxima generación. El operador primario es un crossover que produce dos hijos desde dos padres elegidos. El crossover crea vástagos al intercambiar subárboles entre dos progenitores. Existen otros operadores, tales como: la mutación, la permutación, la edición [94]. Por ejemplo una mutación típica selecciona un nodo en un árbol y genera un nuevo subárbol en forma aleatoria, cuya raíz es el nodo elegido.

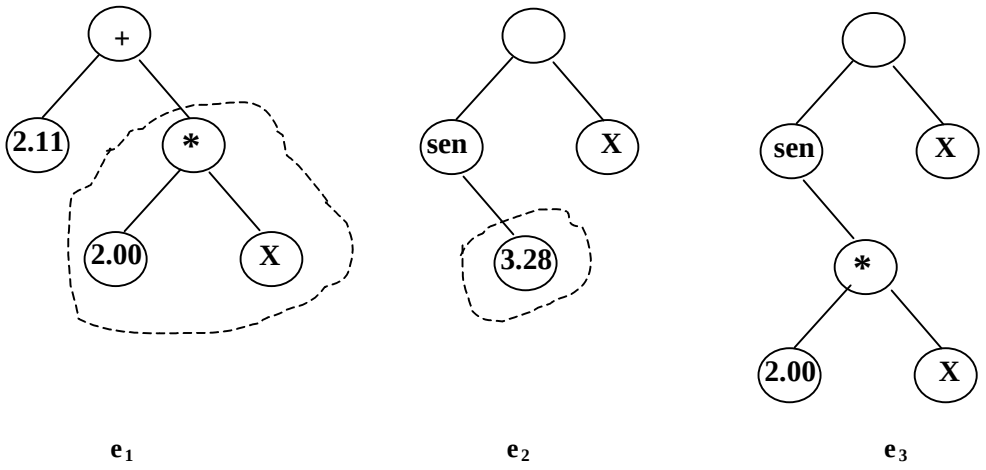

Figura 2.4. Expresión e_3: un hijo de e_1 y e_2. Las líneas de trazos partidos incluyen las áreas a ser intercambiadas durante la operación de crossover.

Koza [96] recientemente ha agregado un paso más para la construcción de un programa genético: agregar un conjunto de procedimientos. Estos procedimientos son llamados Funciones Definidas Automáticamente (ADF en inglés). Este es un concepto extremadamente útil para las técnicas de programación genética, con su mayor contribución en el área de reusabilidad de código. Las ADFs descubren y explotan las regularidades, simetrías, similitudes, patrones, y modularidades del problema en cuestión, y el programa genético final puede llamar a estos procedimientos en diferentes etapas de su ejecución.

El hecho de que la programación genética opera sobre programas de computadoras tiene pocos aspectos interesantes. Por ejemplo, los operadores pueden ser vistos como programas, que pueden sufrir una evolución por separado durante la ejecución del sistema. Adicionalmente, un conjunto de funciones pueden consistir de varios programas que realizan tareas complejas; tales funciones pueden desarrollarse, además, durante la ejecución evolutiva (ej. ADF). Actualmente, esta es una de las áreas de desarrollo de mayor interés en la computación evolutiva y ha acumulado una importante cantidad de datos experimentales (ver por ej. [95, 96]).

CAPÍTULO 3: ALGORITMOS GENÉTICOS

3.1. INTRODUCCIÓN

Un algoritmo genético (AG) es una forma de evolución [133] que ocurre en una computadora. Un algoritmo es una descripción general de un procedimiento, y un programa es su realización como una secuencia de instrucciones a una computadora. Los Algoritmos Genéticos son un método de búsqueda que puede usarse para: resolver problemas y modelar sistemas evolutivos. La idea básica de un algoritmo genético es muy simple. Primero se crea una población de individuos en una computadora (típicamente almacenada como un string binario en memoria), y entonces la población evoluciona usando los principios de variación, selección y herencia.

Los algoritmos genéticos, a diferencia de otras técnicas de búsqueda, realizan una búsqueda multidireccional al mantener una población de múltiples soluciones potenciales. Intentando así, escapar de los óptimos locales. La población sufre una evolución simulada: en cada generación las soluciones relativamente buenas se reproducen, mientras que las soluciones relativamente malas mueren. Los AGs usan reglas de transición probabilísticas para seleccionar individuos a reproducir o a morir.

Con distintas técnicas de transferencia y medidas apropiadas de fitness los Algoritmos Genéticos pueden hacerse a la medida de muchos tipos de problemas, incluyendo optimización de una función o determinación del orden apropiado de una secuencia [81, 130]. Los Algoritmos Genéticos son capaces de explorar un rango mucho más amplio de soluciones potenciales que los convencionales.

Los AGs han dado resultados satisfactorios en la resolución de problemas de optimización como ruteo de redes, scheduling, control adaptativo, game playing, modelado cognitivo, problemas de transporte, problemas del viajente, problemas de control óptimo, optimización de consultas de base de datos [7, 13, 14, 31, 42, 55, 59, 69, 73, 75, 76, 95, 105, 122, 143, 144].

3.2. DESCRIPCIÓN GENERAL DE LOS ALGORITMOS GENÉTICOS (AGS)

Goldberg define a los Algoritmos Genéticos en [69]. Los Algoritmos Genéticos son técnicas estocásticas de búsqueda ciega de soluciones cuasi-óptimas. Ellos mantienen una población que representa a un conjunto de posibles soluciones, llamadas individuos. Tal población es sometida a ciertas transformaciones con las que se trata de obtener nuevos candidatos, y a un proceso de selección sesgado a favor de los mejores individuos.

La principal innovación de los AGs en el dominio de los métodos de búsqueda es la adición de un mecanismo de *selección* de soluciones [113]. La selección presenta dos características: a corto plazo los mejores individuos tienen más probabilidad de sobrevivir, mientras que a largo plazo estos poseen más posibilidades de tener descendencia. A causa de lo anterior, el mecanismo de selección se desdobla en dos operadores:

1. El operador de *selección* elige los elementos a transformarse posteriormente (selección para la reproducción),
2. El operador *reemplazo* elige los elementos que sobreviven (selección para el reemplazo).

En cuanto a las restantes características, cabe destacar que los AGs canónicos o simples son métodos de búsqueda:

- *Ciegos*. Es decir, no disponen de ningún conocimiento específico del problema; de manera que la búsqueda se basa exclusivamente en los valores de la función objetivo.
- *Codificados*. Puesto que no trabajan directamente sobre el dominio del problema, sino sobre representaciones de sus elementos.
- *Múltiples*. Esto es, buscan simultáneamente entre un conjunto de candidatos.
- *Estocásticos*. Referido tanto a las fases de selección como a las de transformación. Ello proporciona control sobre el factor de penetración de la búsqueda.

Todas las características enumeradas se introducen deliberadamente para proporcionar más robustez a la búsqueda, esto es, para darle más eficiencia sin perder la generalidad y viceversa.

Como se menciona en párrafos anteriores los AGs, se diferencian, fundamentalmente, de las técnicas de búsqueda convencionales, porque, comienzan con un conjunto de soluciones llamado *población inicial*, entre otras cosas [63]. Cada *individuo* en la población es llamado *cromosoma*, y representa una solución a un problema dado. Un cromosoma es un conjunto de genes. Un gen puede tomar valores determinados pertenecientes a un conjunto de datos posibles. Dichos valores son llamados *alelos*. En otras palabras un cromosoma es un string de símbolos; que generalmente, pero no necesariamente, es un string de bits. Los cromosomas *evolucionan* a través de sucesivas iteraciones, llamadas *generaciones*. Durante cada generación, los cromosomas son *evaluados*, usando una medida de fitness (o aptitud) [56]. Para crear la próxima población nuevos cromosomas, llamados *hijos*, se forman por medio de:

1. La combinación de dos ó más cromosomas mediante un operador de crossover. Tales cromosomas padres son elegidos en general por su fitness, desde la población actual.
2. La modificación de un cromosoma usando el operador de *mutación*.

Una nueva población es obtenida al:

1. Seleccionar, en general de acuerdo al valor de fitness, algunos de los padres e hijos; y
2. Eliminar a otros para mantener constante el tamaño de la población.

Los cromosomas con mejor fitness tienen más probabilidad de ser seleccionados. Después de varias generaciones, los algoritmos convergen al mejor cromosoma, el cual puede representar la solución óptima o subóptima al problema. La estructura general de los AGs se describe en la figura 3.1. Durante la iteración *t* [107], un AG mantiene una población de soluciones potenciales (cromosomas, vectores), $P(t) = \{x^t_1, \ldots\ldots\ldots\ldots, x^t_n\}$. Cada solución x^t_i es evaluada para obtener una medida de su fitness. Entonces, una nueva población (iteración $t+1$) es producida al seleccionar los individuos de mejor fitness. Algunos miembros de esta nueva población sufren alteraciones por medio del crossover y la mutación, y

así forman nuevas soluciones. El crossover combina las características de dos cromosomas padres para formar dos vástagos similares al intercambiar los segmentos correspondientes de los padres. La idea que trae aparejada la aplicación del crossover es el intercambio de información entre distintas soluciones potenciales. Este es el principal operador genético en los AGs.

```
procedure Genetic Algorithm
begin
        t ← 0;
        inicializar P(t);
        evaluar estructuras en P(t);
        while la condición de terminación no se satisfaga do
        begin
          t ← t + 1;
          select_repro C(t) desde P(t-1);
          recombinar y mutar estructuras en C(t) formando C'(t);
          evaluar estructuras en C'(t);
          select_replace P(t) desde C'(t) y P(t-1);
        end
end
```

Figura 3.1. Estructura General de los Algoritmos Genéticos

La operación de mutación altera en forma arbitraria uno o más genes de un cromosoma seleccionado, al realizar un cambio aleatorio con una probabilidad establecida para la mutación. La mutación introduce una leve variabilidad en la población.

Un algoritmo genético para un problema en particular posee los siguientes cinco componentes:

1. Una representación genética para soluciones potenciales al problema.
2. Una manera de crear una población inicial de soluciones potenciales.
3. Una función de evaluación que cumple el papel del medio ambiente, ordenando las soluciones en términos de sus fitness,
4. Operadores genéticos que alteran la composición de los hijos.
5. Valores para los distintos parámetros que usa un algoritmo genético, como por ejemplo: tamaño de la población, probabilidades de aplicar los operadores genéticos, etc.

3.2.1. EL GENOTIPO

En genética, el *genotipo* es la colección abstracta de genes poseídos por un individuo [119, 136]. La estructura que contiene a los genes es el *cromosoma*. Un gen tiene un valor (*alelo*) y una posición en el genotipo (*locus*). En la naturaleza existe una compleja traducción entre el genotipo y las propiedades del organismo (el *fenotipo*). Se ha observado que un único gen puede afectar múltiples cualidades en el fenotipo, y esto se denomina *pleiotropía*. La otra traducción que se observa es la siguiente: múltiples genes controlan o afectan a una sola cualidad o característica del fenotipo (*poligenia*). Los sistemas evolutivos naturales hacen un uso extensivo de la poligenia y la pleiotropía [55]. Típicamente los AGs se abstraen mucho de la riqueza de la evolución natural y no explotan las codificaciones y mecanismos de crossover complejos que se encuentran en la naturaleza [55]. Muchos ejemplos pueden encontrarse en la literatura de los AGs, donde los genes tienen una traducción uno a uno a las propiedades fenotípicas. Este tipo de traducción representa una simplificación de la evolución natural.

Por ejemplo, una codificación binaria es una excelente opción para problemas en los cuales un punto del problema se traslada naturalmente a un string de ceros y unos. Un string binario se usa, a menudo, para representar tal codificación, ej. $E = [10001010]$, donde la codificación E tiene 8 genes. La posición (o locus) del i-*ésimo* gen es simplemente el bit i en el *bitstring*, y el valor o alelo está dado por el valor del bit i $E[i]$.

El problema de satisfacción booleana (SAT problem, en inglés) [32] se traduce muy bien a un string binario, ya que una solución a un problema SAT es una asignación de variables binarias. Sin embargo, para muchos problemas una codificación binaria no es apropiada porque:

1. Un alto grado de *epistasis* significa que buenos segmentos (bloques) no se pueden formar, entonces el problema es *deceptivo*. La epistasis mide la importancia en la cual la contribución de fitness de un gen depende de los valores de otros genes.

2. *La representación natural* del problema requiere un conjunto de símbolos de mayor orden que el binario. En estos casos un alfabeto de orden mayor puede ser más eficiente.
3. *Las soluciones ilegales* con codificación binaria pueden ser introducidas por los operadores genéticos. Ya que tal codificación puede no describir naturalmente un punto del problema.

3.2.2. EL FENOTIPO

En genética, el fenotipo es un rasgo distintivo o una característica medible que posee un organismo [119]. En AGs, el fenotipo indica las características definitorias o las cualidades del genotipo entero. En algunas situaciones, los rasgos fenotípicos surgen directamente desde el genotipo. Por ejemplo, si el genotipo codifica a un único parámetro binario de un problema, entonces el punto de dicho problema es descripto fenotípicamente al decodificar este parámetro binario en una cantidad numérica. Esta traducción directa entre el genotipo y el fenotipo es evidente en el problema del viajante de comercio (TSP en inglés) con una codificación de permutación [117, 118]. Sin embargo en otros tipos de problemas las características fenotípicas no surgen de forma tan sencilla; es el caso de una representación de *job shop scheduling* [23].

3.2.3. EXPLOTACIÓN VERSUS EXPLORACIÓN

La búsqueda es uno de los métodos más comunes para resolver problemas, en los cuales no se puede determinar a priori una secuencia de pasos para llegar a una solución. Dos puntos importantes son abordados por las estrategias de búsquedas, ciegas o heurísticas, ellos son: la *explotación* de la mejor solución y la *exploración* del espacio de búsqueda [15]. Michalewicz [107] hace una comparación entre las técnicas de búsqueda *hill-climbing, simulated annealing* y los algoritmos genéticos. Hill-climbing es una estrategia que explota la mejor solución para posibles mejoras pero ignora completamente la exploración del espacio de búsqueda. En tanto que simulated annealing [1] permite la exploración

del espacio de búsqueda a altas temperaturas, donde cualquier movimiento es permitido, y la explotación de la mejor solución cuando la temperatura es baja. Mientras que los algoritmos genéticos, son una clase de métodos de búsqueda de propósito general que combinan elementos de la búsqueda estocástica y dirigida; las cuales pueden remarcar el balance entre la exploración y la explotación del espacio de búsqueda.

En los AGs, los operadores de crossover, también llamados de cruce, se encargan de explotar las mejores características que disponga la población actual. Mientras que los operadores de mutación se encargan de explorar los nuevos dominios en busca de mejores soluciones [113]. Desde esta perspectiva, los operadores de cruce son los que principalmente se encargan de la búsqueda en profundidad (explotación) y los de mutación de la búsqueda en amplitud (exploración). Así dando mayor importancia a unos que a otros es posible ajustar el tipo de búsqueda a las necesidades del problema.

El crossover se encarga de realizar un intercambio estructurado de información, mientras que la mutación proporciona una garantía de accesibilidad para todos los puntos del espacio de búsqueda.

3.2.4. LIMITACIÓN EN LOS AGS

El aspecto más visible de la debilidad es la *ineficiencia*: los métodos débiles de resolución de problemas presentan las ventajas de ser muy poco específicos, poco exigentes y notablemente eficaces [113]. Sin embargo, todos ellos son ineficientes en mayor o menor grado, y especialmente si se los compara con otros métodos heurísticos. Los AGs sólo se pueden considerar eficientes en comparación con otros métodos estocásticos de búsqueda ciega, pero se puede garantizar que si se encuentra un método heurístico para resolver un problema específico este siempre lo hará mucho más eficientemente que un AG.

Pero eso no es todo, otro de los inconvenientes prácticos de la debilidad es la tendencia al *extravío* en la búsqueda. Este fenómeno es el siguiente: el AG simple realiza la búsqueda de los mejores puntos utilizando únicamente la aptitud (o fitness) de los individuos para luego recombinarlos. En ocasiones ocurre que la

información proporcionada (ej. el fitness) resulta insuficiente para orientar correctamente al algoritmo en su búsqueda del óptimo. A ésto se lo llama *desorientación* (deception en inglés). Esto se concreta porque ciertas combinaciones válidas de buenos segmentos originan individuos de baja aptitud. En el peor de los casos puede ocurrir que este fenómeno sea tan fuerte como para impedir que el AG converja al óptimo global, desviándolo finalmente hacia un óptimo local. Esto es, el extravío propiamente dicho. Afortunadamente, aunque puede que un AG se desoriente no necesariamente esto ocurre; para eso es necesario partir de una mala población inicial o plantear un problema especialmente diseñado para que se extravíe.

En resumen, para que funcione correctamente el mecanismo básico de los AGs es necesario proporcionarle una mínima cantidad (y calidad) de información. Si no se proporciona ese mínimo el mecanismo no funciona con propiedad, y el AG evolucionará incorrectamente.

Como es de suponer, el hecho que más contribuye a la existencia de desorientación es la forma de la función de fitness. Típicamente, la desorientación es frecuente en problemas donde los puntos óptimos están asilados y rodeados de puntos de muy baja aptitud. Sin embargo, conviene señalar que la presencia de desorientación suele estar fuertemente estimulada por una mala codificación que 'oculte' la ya de por sí escasa información disponible. De modo recíproco una buena codificación reduce la probabilidad de extravío.

Un caso especialmente desfavorable ocurre cuando hay una fuerte interacción entre dos o más atributos (ej. genes), de tal forma que la contribución al fitness de un individuo que realiza cierto gen depende, en gran medida, de los valores que tomen otros. A este fenómeno se lo denomina *epistasis* o acoplamiento y su presencia garantiza la desorientación dado que en esas condiciones resulta muy difícil constituir buenos segmentos.

3.2.5. EL PROBLEMA DE LA DIVERSIDAD EN LOS AGS

A efectos prácticos, es fundamental para el buen funcionamiento de un AG tener controlada en todo momento la diversidad de la población. Se entiende a la

diversidad en sentido general como 'diversidad de individuos' y en particular como 'diversidad de aptitudes'.

Con poca diversidad de individuos hay poca variedad de segmentos, a causa de ello el operador de crossover pierde casi por completo la capacidad de intercambio de información útil entre individuos, y en definitiva, la búsqueda se estanca. La necesidad de tener controlada la diversidad de aptitudes radica en la imposibilidad práctica de trabajar con una población infinita.

No es conveniente tener poca diversidad de aptitudes ya que en tal caso todos los individuos tendrían más o menos las mismas posibilidades de sobrevivir, la selección reproduciría la situación anterior y todo el peso de la búsqueda recaería en los operadores genéticos, lo que a la larga sería poco más que una búsqueda aleatoria. Como resultado final la búsqueda se estanca y, la situación puede empeorar si la población es finita. Resumiendo, para que la selección sea efectiva, la población debe contener en todo momento una cierta variedad de aptitudes.

Por otra parte cuando la población es finita –es decir, siempre- tampoco se puede tener gran disparidad de aptitudes, pues ello suele afectar muy negativamente a la diversidad de la población.

3.2.5.1. CONVERGENCIA PREMATURA POR PROBLEMAS CON LA DIVERSIDAD

Sea un AG básico con una población finita y admítase por simplicidad que la función de fitness coincide con la función de evaluación. En algún momento puede ocurrir que un individuo o un grupo de ellos obtengan un fitness notablemente superior a los demás. Esto es, especialmente probable en las fases tempranas de la evolución; donde suele surgir un buen candidato desde una población de individuos mediocres, por aplicación de los operadores genéticos. En tal circunstancia existe el riesgo de que se produzca una *evolución en avalancha*; debido al incremento de la presencia en la población de individuos más aptos, que por ser finita disminuye su diversidad. Ello hace que en la siguiente generación se favorezca aún más a los individuos más aptos hasta que éstos acaban dominando por completo la población. A esto se le conoce como el

fenómeno de los *superindividuos*. Habitualmente ocurre que tales superindividuos sólo son los más aptos en cierto momento, pero no los potencialmente más aptos -es necesario tener en cuenta que la falta de diversidad estanca la búsqueda-; provocando, en general, una *convergencia prematura* del AG, habitualmente hacia un subóptimo.

La única circunstancia en que este fenómeno es tolerable e incluso deseable, es en las fases tardías de la evolución; cuando el algoritmo genético ha 'localizado' correctamente la zona del espacio de búsqueda donde la solución óptima se encuentra, pero nunca antes.

La posibilidad de convergencia en avalancha es un fenómeno inherente a los AGs con población finita, debido a que la finitud activa el bucle "selección sesgada versus incremento de la diversidad". Tanto es así, que dicho fenómeno puede ocurrir incluso cuando no haya diferencias apreciables en las aptitudes de la población. Ya se ha comentado que en tal circunstancia la búsqueda se paraliza, dado que la selección no concede mayor preeminencia a uno u otro individuo. Sin embargo, cuando la población es finita y especialmente cuando no es muy grande, puede ocurrir el siguiente fenómeno conocido como *deriva genética*: en una población finita los procesos de muestreo son siempre imperfectos, pudiendo favorecer más de lo que les corresponde a los individuos ocasionalmente más aptos; a esto se lo denomina *presión selectiva*. Debido a la finitud, especialmente, en poblaciones de tamaño pequeño y como consecuencia de esos errores estocásticos en los muestreos, se puede producir la pérdida de material genético irrecuperable; haciendo que el AG converja prematuramente hacia un "individuo afortunado".

Es imprescindible tener el control sobre la diversidad de aptitudes de la población (finita) para evitar que se produzca una convergencia prematura (avalancha) ya sea por la presencia de superindividuos o incluso por deriva genética.

Mecanismos para prevenir la convergencia prematura son desarrollados en [40] y en [2].

3.3. REPRESENTACIÓN GENÉTICA (CROMOSOMAS)

Uno de los puntos clave de los AGs es como codificar la solución de un problema en un cromosoma. El trabajo de Holland ha sido llevado a cabo usando strings binarios (AG simple). Para muchas aplicaciones de AGs, especialmente para los problemas de la ingeniería industrial, el AG simple es difícil de aplicar porque el string binario no es una codificación natural. Han sido varias las codificaciones creadas con un formato diferente al del string, algunas de ellas son: codificación entera creada para problemas de optimización con restricciones, codificación en números reales para problemas de optimización de funciones reales. Elegir una representación apropiada de soluciones candidatas al problema en cuestión es fundamental para aplicar AGs. En cualquier aplicación es necesario realizar un análisis cuidadoso, para asegurar una apropiada representación de soluciones junto con operadores genéticos específicos del problema.

Una de las características básicas de los AGs, es que trabajan sobre un espacio codificado y un espacio de soluciones en forma alternativa. Esto es, las operaciones genéticas trabajan sobre un espacio codificado (cromosomas, genotipo), mientras que las operaciones de evaluación y selección lo hacen sobre el espacio de soluciones (fenotipo). Para las representaciones que no usan strings, surgen puntos críticos con la codificación y decodificación entre cromosomas y soluciones (o la traducción entre fenotipo y genotipo):

- *La factibilidad de un cromosoma*.
- *La legalidad de un cromosoma*.
- *La unicidad de la traducción.*

3.3.1. FACTIBILIDAD DE UN CROMOSOMA

La factibilidad de un cromosoma, se refiere al fenómeno de que una solución decodificada a partir de un cromosoma se encuentre en la región factible de un problema dado. La no factibilidad de un cromosoma proviene de la naturaleza de los problemas de optimización con restricciones. Todos los métodos, ya sean los convencionales o los evolutivos, deben manejar restricciones. En muchos

problemas de optimización, las regiones factibles pueden ser representadas como un sistema de ecuaciones o inecuaciones (lineales o no). Para tales casos, muchos métodos eficientes de penalidad se han propuesto para manejar cromosomas no factibles en los AGs [62, 106, 131]. En los problemas de optimización con restricciones, el óptimo típicamente ocurre en los límites entre las áreas factibles y las no factibles. La penalidad fuerza a la búsqueda genética para que se aproxime al óptimo desde ambas regiones.

3.3.2. LEGALIDAD DE UN CROMOSOMA

La legalidad de un cromosoma, se refiere al fenómeno de que un cromosoma represente una solución a un problema dado. La ilegalidad del cromosoma se origina en la naturaleza de la técnica de codificación. Para muchos problemas de optimización combinatorial se usan codificaciones específicas al problema, con las cuales se producen vástagos ilegales si se aplica la operación de crossover básica de un punto. Ya que, un cromosoma ilegal no puede decodificarse en una solución, en consecuencia el cromosoma no puede evaluarse; es decir que la penalidad no es aplicable en esta situación. Técnicas reparadoras se usan generalmente para convertir un cromosoma ilegal en uno legal. Por ejemplo el conocido operador PMX (el cual se introduce en el capítulo 5) es, esencialmente, un tipo de crossover de dos puntos para representaciones por permutación junto con un procedimiento para resolver la ilegalidad causada por un simple crossover de dos puntos. Orvosh y Davis [112] muestran que para muchos problemas de optimización combinatorial, es relativamente fácil reparar un cromosoma ilegal o no factible.

3.3.3. UNICIDAD DE UN CROMOSOMA

La función de traducción (mapping en inglés) desde el cromosoma a las soluciones (decodificación) puede pertenecer a uno de los siguientes casos:

1º. Traducción 1 a 1.

2º. Traducción *n* a 1.

3º. Traducción 1 a *n*.

En el primer caso es el mejor de los tres, mientras que el tercero es el menos deseado. Necesitamos considerar estos problemas cuidadosamente cuando se diseña una nueva codificación no binaria para construir un algoritmo genético efectivo.

3.4. SELECCIÓN

El principio que secunda a los algoritmos genéticos es, fundamentalmente, la selección natural Darwiniana. La selección impone la dirección en el proceso de búsqueda, donde la presión en la selección es crítica. En un extremo, la búsqueda termina prematuramente; mientras que en el otro, el progreso puede ser más lento de lo necesario. En general una baja presión selectiva se sugiere al inicio de la búsqueda genética, para favorecer la exploración del espacio de búsqueda; mientras que una alta presión selectiva se recomienda al final para explotar las regiones más prometedoras del espacio de búsqueda.

Una mayor presión selectiva restringe la cantidad de individuos a recombinar. Por ende se pierde diversidad poblacional (o diversidad de individuos). La pérdida de diversidad poblacional, el número limitado de generaciones y el tamaño de la población son causas de la convergencia prematura a un óptimo local antes de encontrar el óptimo global o un valor cercano al mismo.

Un parámetro asociado a la presión selectiva es el *takeover time*, que se define en el trabajo de Goldberg y Deb [71], como el número de generaciones necesarias para que el mejor individuo encontrado en la población inicial cope la población al aplicar repetidamente sólo un determinado mecanismo de selección. Si el takeover time es grande entonces la presión selectiva de un operador de selección es débil, si es pequeño ocurre lo contrario.

La *diversidad poblacional* fue introducida por Bäck y Hoffmeister [5], en términos de la medida del bias definida por Grefenstette:

$$b(P(t)) = \frac{1}{l \cdot \mu} \sum_{j=1}^{l} max \left(\sum_{\substack{i=1 \\ a_{i,j}^t = 0}}^{\mu} (1 - a_{i,j}^t), \sum_{\substack{i=1 \\ a_{i,j}^t = 1}}^{\mu} a_{i,j}^t \right)$$

donde l es el largo del cromosoma y $a^t_{i,j}$ denota el valor del alelo. El bias b $(0.5 \leq b \leq 1.0)$ indica el porcentaje promedio de mayor valor en cada posición del individuo. Valores pequeños de b indican una alta diversidad genotípica y viceversa. El bias b puede ser usado para formular un adecuado criterio de finalización.

La selección dirige la búsqueda genética hacia regiones prometedoras en el espacio de búsqueda. Los principales tópicos que involucra esta fase son:

- El espacio de muestreo.
- Los mecanismos de muestreo.
- La probabilidad de selección.

3.4.1. ESPACIO DE MUESTREO

El procedimiento de selección puede crear una nueva población para la próxima generación basada en cada uno de los padres y los hijos o partes de ellos. Un espacio de muestreo es caracterizado por dos factores: *tamaño* e *integrantes* (padres o hijos). Donde μ denota el tamaño de la población y λ denota la cantidad de hijos producidos en cada generación. Un espacio de muestreo regular tiene el tamaño μ y contiene a todos los hijos pero sólo a una parte de los padres [30, 77, 82, 107]. El espacio de muestreo extendido tiene el tamaño de $\mu + \lambda$ y contiene el total de padres y de hijos [5, 6, 53, 128].

3.4.1.1. ESPACIO DE MUESTREO REGULAR

En los algoritmos genéticos simples, los padres son reemplazados por sus hijos. Esto es denominado *reemplazo generacional*. Con esta estrategia, se pueden perder muy buenas soluciones en el proceso evolutivo. Ya que los vástagos pueden ser peores que sus progenitores. Para superar este problema, se han

propuesto distintas *estrategias de reemplazo*. Holland sugiere que cuando un hijo nace, reemplace a un cromosoma de la población actual; elegido aleatoriamente [82]. De Jong propone una *estrategia de crowding* [30]. En el modelo de crowding, cuando un hijo nace, un padre es selleccionado para morir. Estos padres, son aquellos que más se asemejan al nuevo hijo. Esta semejanza se mide calculando la distancia Hamming entre los cromosomas.

En el trabajo de Holland, se lleva a cabo la selección de los individuos (padres) que van a ser recombinados; se forma una nueva población al reemplazar a los padres por sus hijos. Esto se denomina *plan reproductivo*.

En el trabajo de Grefenstette y Baker [77], se usa la selección para formar la próxima generación, generalmente con un mecanismo probabilístico.

Michalewicz da una descripción detallada de algoritmos genéticos simples donde los vástagos en cada generación reemplazan a sus padres; la próxima generación se forma por medio de una selección *roulette wheel* (proporcional) [107]

La figura 3.2 ilustra la selección basada en un espacio de muestreo regular.

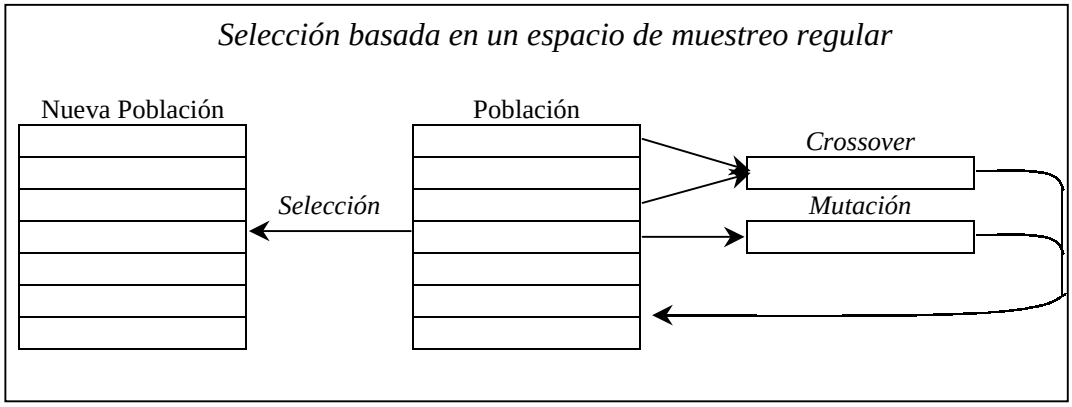

Figura 3.2. Selección realizada sobre un espacio de muestreo regular

3.4.1.2. ESPACIO DE MUESTREO EXTENDIDO

Cuando una selección se realiza sobre un espacio de muestreo extendido, tanto padres como hijos tiene la misma oportunidad por sobrevivir. Un caso típico es la selección (μ + λ) [53], usada originalmente en las estrategias evolutivas [128]. Bäck y Hoffmeister la introducen en los AGs [5, 6]. Con esta estrategia μ padres y λ hijos compiten por sobrevivir y los μ mejores son seleccionados como padres

en la próxima generación. Otro caso proveniente de las estrategias evolutivas, es la selección (μ, λ); en la cual se eligen los μ mejores hijos como padres de la próxima generación ($\mu < \lambda$). Ambos métodos son totalmente determinísticos y pueden convertírselos en probabilísticos.

Aunque la mayoría de los métodos se basan en un espacio de muestreo regular, es fácil implementar uno extendido. La figura 3.3 muestra la selección basada en un espacio de muestreo extendido.

Una ventaja evidente de esta opción es que se puede mejorar la performance del AG al incrementar las probabilidades de crossover y mutación. Tampoco es necesario preocuparse por la gran perturbación que pueden ocasionar estas altas probabilidades si la selección se realiza sobre un espacio de muestreo extendido. Ya que existe una probabilidad no nula de supervivencia de los padres.

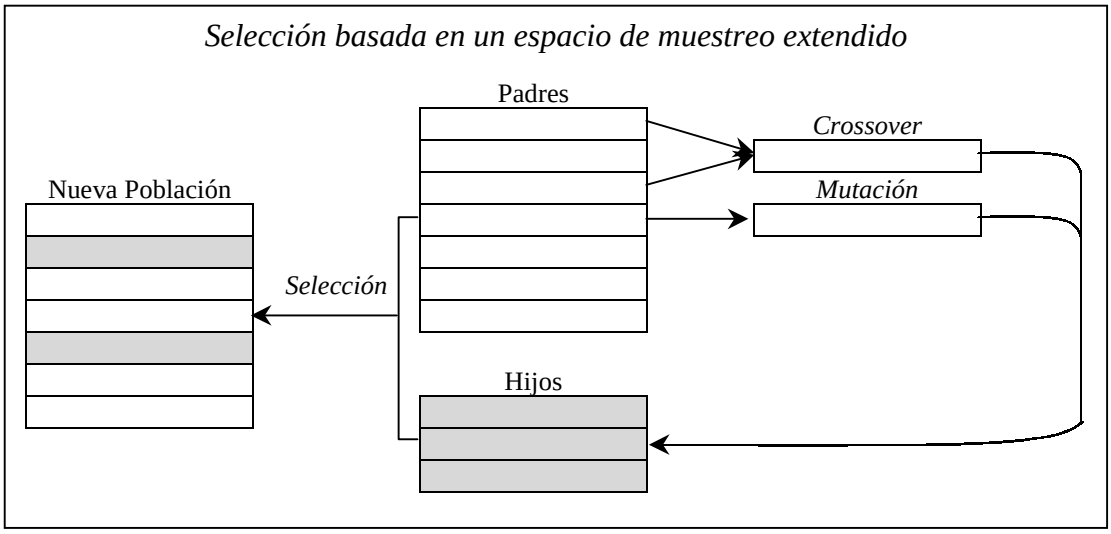

Figura 3.3. Selección realizada sobre un espacio de muestreo extendido

3.4.2. MECANISMOS DE SELECCIÓN

Los mecanismos de selección encierran el problema de cómo elegir cromosomas desde el espacio de muestreo. Las tres alternativas básicas usadas para el muestreo de individuos son:

- El muestreo estocástico.
- El muestreo determinístico.
- El muestreo mixto.

3.4.2.1. MUESTREO ESTOCÁSTICO

Una característica común en esta clase, dada por Baker, es que la fase de selección determina el número actual de copias que cada cromosoma va a recibir basado en la probabilidad de sobrevivencia (o de selección) [11]. Es decir que la fase de selección está compuesta por dos partes:

1. La determinación del valor esperado del cromosoma.
2. La conversión de los valores esperados al número de hijos.

El valor esperado de un cromosoma, es un número real que indica el número medio de vástagos que un cromosoma debería recibir. El procedimiento de muestreo es usado para convertir un valor esperado real al número de hijos.

El método mejor conocido de esta clase es la *selección proporcional* (de Holland) o *selección de la ruleta*. La idea básica es determinar la *probabilidad de selección* para cada cromosoma proporcionalmente a su valor de fitness. Para el cromosoma a_i con fitness $f(a_i)$, su probabilidad de selección p_{sel} es calculada como sigue:

$$P_{sel}\, f(a_i) = \frac{f(a_i)}{\sum_{j=1}^{\mu} f(a_j)}$$

Los individuos son ubicados en segmentos continuos de una recta, de forma tal que el segmento correspondiente a cada individuo es proporcional en tamaño a su fitness. Se genera un número aleatorio y se selecciona el individuo cuyo segmento alcanza dicho número. El proceso se repite hasta que se obtiene el número deseado de individuos (matting pool).

Baker [10, 11] propone el *muestreo estócastico universal* (conocido en inglés como Stochastic Universal Sampling – SUS). Aquí también, los individuos son ubicados en segmentos continuos sobre una línea, de manera tal que el segmento correspondiente a cada individuo es igual en tamaño a su fitness. Los punteros a cada segmento están separados en distancias iguales sobre la línea dependiendo de la cantidad de individuos a ser seleccionados. Si se considera N el número de individuos a seleccionarse, entonces la distancia entre los punteros es de $1/N$, la

posición del primer puntero se genera aleatoriamente dentro del rango [0, 1/*N*]. SUS asegura una selección de hijos más exacta que el método anterior. La figura 3.4 ilustra este método.

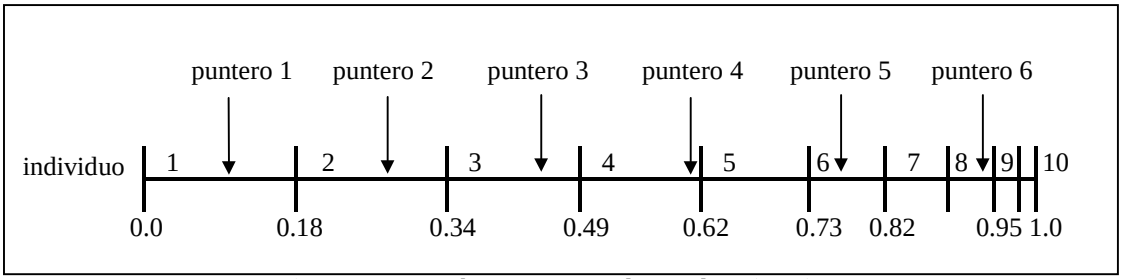

Figura 3.4. Stochastic Universal Sampling (SUS).

La consideración básica de SUS es mantener el número esperado de copias de cada cromosoma en la próxima generación. Existen dos razones para usar esta estrategia:

1. Previene la aparición de super cromosomas desde la población dominante al mantener demasiadas copias de estos en la misma. Esto es causa de convergencia prematura.
2. Mantiene la diversidad poblacional ya que el matting pool puede contener mucha más información para la búsqudea genética.

3.4.2.2. MUESTREO DETERMINÍSTICO

Selecciona los mejores μ cromosomas desde el espacio de muestreo. Los métodos de selección $(\mu+\lambda)$ y (μ,λ) hacen esto [6]. Ambas prohiben los cromosomas duplicados durante la selección.

La *selección por truncamiento* y la *selección por bloques* pertenecen a esta clase, y ordenan a todos los cromosomas de acuerdo a su fitness y selecciona a los mejores como padres [140]. En la selección por truncamiento un umbral *U* es definido tal que los *U*% mejores cromosomas son seleccionados y cada uno recibe $100/U$ copias. La selección por bloques es similar a la anterior, si para un tamaño de población dado μ, se da simplemente *s* copias a los mejores μ/s cromosomas. Ambas implementaciones son idénticas cuando $s = \mu/U$.

La *selección elitista* asegura que el mejor cromosoma se pase a la nueva generación si este no fuese elegido a través de otro proceso de selección.

El reemplazo generacional puede ser visto como otra versión de la opción determinística. Donde el conjunto total de padres se reemplaza por el de sus hijos [107, 137, 147].

3.4.2.3. MUESTREO MIXTO

Esta opción contiene en forma simultánea características aleatorias y determinísticas. Un ejemplo típico es la *selección por torneo* dada por Goldberg [70]. Este método aleatorio elige un conjunto de cromosomas (el tamaño de este conjunto se denomina *tamaño del torneo*) y toma el mejor de ellos para formar el conjunto de padres (matting pool) para la reproducción. Un tamaño de torneo común es 2, y se denomina *torneo binario*.

En la *selección por torneo estocástico* sugerida por Whetzel [145], las probabilidades de selección se calculan y pares sucesivos de cromosomas se eligen por medio de un método de selección proporcional o uniforme. Después de lo cual el cromosoma con fitness más alto se inserta en la nueva población. El proceso continúa hasta que la población está completa.

En el *muestreo estocástico remanente* propuesto por Brindle [16], cada cromosoma se muestra de acuerdo a la parte entera del número esperado. Compitiendo por las posiciones restantes en la población, de acuerdo a la parte fraccionaria del número esperado.

3.4.3. PROBABILIDAD DE SELECCIÓN

En este punto se trata cómo determinar la probabilidad de selección para cada cromosoma. La *probabilidad de selección* es un parámetro importante de un mecanismo de selección y normalmente determina el número de copias esperadas de un individuo después de la selección.

En un procedimiento de selección proporcional, la probabilidad de selección es proporcional a su fitness. Este esquema simple exhibe algunas propiedades a

considerar. Por ejemplo, en generaciones tempranas existe la tendencia de que unos pocos super cromosomas dominen el proceso de selección; en generaciones más tardías, cuando la población es convergente, la competencia entre cromosomas es más débil y la búsqueda se convierte en aleatoria.

Los mecanismos de selección por *escalamiento* y *ranking* son propuestos para mitigar estos problemas. El primero convierte a los valores de la función objetivo en algunos valores reales positivos, y la probabilidad de selección para cada cromosoma es determinada de acuerdo a estos valores. El método de selección por ranking ignora los valores de la función objetivo actual y usa un ordenamiento de cromosomas en lugar de determinar una probabilidad de sobrevevivencia.

El escalamiento de fitness tiene una doble intención:

1. Mantener una diferencia razonable entre los valores de fitness relativos.
2. Prevenir un rápido copamiento de la población por unos pocos super individuos, al limitar la competencia en las primeras generaciones y estimularla más tarde.

La mayoría de los métodos de escalamiento usan parámetros dependientes del problema. El ordenamiento (ranking) de fitness tiene un efecto similar al escalamiento de fitness pero evita la necesidad de escalamiento extra [115].

El escalamiento de valores de la función objetivo ha sido ampliamente aceptado en la práctica y una amplia variedad de mecanismos de escalamiento se han propuesto. Goldberg [69] y Michalewicz [107] han hecho una buena compilación de todos ellos.

3.5. MUTACIÓN

El operador de mutación para los algoritmos genéticos ha sido introducido por Holland como un operador secundario que ocasionalmente cambia un único bit del individuo al invertirlo [82]. La probabilidad de mutación p_m, por bit, pertenece al intervalo [0, 1] y es usualmente muy baja en los AGs. Algunas configuraciones comunes son $p_m = 0.001$ [30], $p_m = 0.01$ [74] y p_m en el intervalo [0.005, 0.01] en [121]

La forma del operador de mutación, $m'_{\{p_m\}} : S \to S$, usando la inversión del bit por cada posición que sufre mutación, produce un string de bits $\vec{a}' = (a'_1,....,a'_l) = m'_{\{p_m\}}(\vec{a})$ de acuerdo a:

$$a'_i = \begin{cases} a_i, & \text{si } x_i > p_m \\ 1 - a_i, & \text{si } x_i \le p_m, \end{cases} \qquad \forall i \in \{1,....,l\}$$

donde $x_i \in [0,1]$.

Bajas probabilidades de mutación, garantizan que un individuo producido por mutación no difiera genéticamente demasiado de sus ancestros.

3.6. RECOMBINACIÓN

Mientras que la mutación en los algoritmos genéticos sirve como un operador para introducir alelos perdidos en la población, el operador de crossover es el más importante en los GAs [9]. Mediante la recombinación, segmentos útiles de diferentes padres se combinan para obtener un nuevo individuo beneficiado con la combinación de bits de sus padres. De esta manera, se espera que substrings más grandes de alto fitness surjan, arribando finalmente a una buena solución.

El crossover en los algoritmos genéticos es siempre un operador sexual $r'_{\{p_c\}} : S^2 \to S$ que con probabilidad p_c selecciona dos padres $\vec{s}$ y $\vec{v}$, y las recombina para formar dos nuevos individuos. Las probabilidades de crossover propuesta son $p_c = 0.6$ [80], $p_c = 0.95$ [74] y $p_c \in [0.75,0.95]$ en [121]. El tradicional *crossover de un punto* introducido por Holland elige aleatoriamente una posición de corte $j \in \{1,...,l-1\}$ dentro del cromosoma e intercambia los genes a la derecha de esta posición entre ambos individuos [82], resultando en:

$$\vec{s}' = (s_1,....s_{j-1}, s_j, v_{j+1},...., v_l)$$

$$\vec{v}' = (v_1,....v_{j-1}, v_j, s_{j+1},...., s_l)$$

Este operador de crossover tiene una performance claramente inferior a la de otros operadores de crossover. El crossover de un punto sufre una fuerte dependencia de la probabilidad de intercambio sobre las posiciones de los bits.

Existe una asimetría entre la aplicación del crossover y la mutación: la unidad básica de la mutación es el gen, la del crossover es el individuo; en consecuencia, la probabilidad de que un individuo se cruce no depende de su longitud, mientras que la probabilidad de que contenga genes mutados es más alta cuanto más largo sea.

A continuación se describen los tipos de crossover más comunes:

- *Crossover Multipunto* [39]*:* consiste en combinar los individuos en torno a dos o más puntos de corte. Este cruce mejora la capacidad de procesado de segmetnos, pero a costa de perder velocidad de convergencia.
- *Crossover Segmentado* [39]: Es una versión del crossover multipunto que permite la introducción de variabilidad en el número de puntos de corte. Consiste en reemplazar el número fijo de los puntos de corte con una probabilidad de segmentación p_{seg} que da cuenta de la posibilidad de que se produzca un cruce al llegar a cierto punto del string. Así se trata de corregir la asimetría con respecto a la mutación.
- Por ejemplo, si la probabilidad de segmentación es $p_{seg} = 0.20$ entonces el promedio esperado de cruces será $0.2 \cdot l$ en cada par de progenitores, aunque no necesariamente tomará ese valor.
- *Crossover Uniforme* [137]: para cada bit del primer descendiente se decide, con probabilidad p_{unif}, de qué progenitor heredará su valor en esa posición. El segundo descendiente recibe el correspondiente gen del otro progenitor. El crossover uniforme se usa para combinar atributos específicos, independientemente de la posición en que han sido codificados. No obstante se recomienda utilizar el crossover uniforme sólo en casos concretos en los que haya motivos fundados para hacerlo.

CAPÍTULO 4: ALGORITMOS EVOLUTIVOS AVANZADOS

4.1. INTRODUCCIÓN

En los Algoritmos Evolutivos (AEs) basados en algoritmos genéticos vistos hasta el momento, el mecanismo de selección busca individuos en el espacio del problema al usar el principio Darwiniano de la selección natural y la supervivencia del más fuerte [43]. Básicamente los Algoritmos Evolutivos simulan la evolución (adaptación natural) de una población de soluciones para un problema dado. Después de crear la población inicial, el proceso evolutivo de un algoritmo genético consiste en:

1. la evaluación de todos los individuos en la población,
2. la selección de una nueva población intermedia (los mejores individuos tienen mayores oportunidades de ser elegidos), y
3. el intercambio de su código genético.

Estas tres etapas se reiteran hasta que alguna condición de terminación se satisfaga.

Los AEs basados en algoritmos genéticos involucran la selección de dos individuos padres para crear a lo sumo dos cromosomas hijos. Inspirados en la naturaleza, Eiben [35], Esquivel [43], y otros han creado distintos algoritmos evolutivos avanzados, de dónde emergen reformas de los mecanismos de selección y crossover. Esto surge como una necesidad de mejorar la performance de los AEs en problemas de alta complejidad. Tales AEs avanzados mitigan el problema de la debilidad, el de la diversidad genética y el de la convergencia prematura (capítulo 2). Se describen algunos de ellos en las siguientes secciones.

4.2. ALGORITMOS EVOLUTIVOS CON RECOMBINACIÓN DE MÚLTIPLES PADRES

Es un hecho de la naturaleza que la creación de nuevos individuos siempre ocurra a través [35] de una reproducción asexual (un padre) o sexual (dos padres). Sin embargo para un matemático/a o para un informático/a ésta no es una restricción. Eiben [35] ha estudiado cuando un crossover de múltiples padres

debiese ser evitado por razones prácticas o cuando este ofrece ventajas que no se han usado. Para lo cual Eiben entre otros han descubierto la necesidad de crear nuevos métodos de crossovers, para intercambiar la información genética de más de dos padres. Los mecanismos básicos de recombinación, para este caso son: el *scanning* de genes [34], el crossover diagonal [36], y el crossover basado en la adyacencia.

Eiben concluye, en función a los experimentos realizados, que los algoritmos evolutivos con recombinación de múltiples padres incrementan su performance; en comparación con los que aplican crossover de dos padres. Pero si bien en algunos casos de prueba, el aumento de la cantidad de padres significa una mejora sustancial en los resultados; esto no es generalizado.

4.2.1. SCANNING CROSSOVER (SCANNING DE GENES)

La técnica de scanning genera un solo hijo de *n* padres. El mecanismo general para el scanning es asignar un puntero a cada padre y a cada hijo. El puntero del hijo atraviesa todas sus posiciones de izquierda a derecha, una por vez. En cada paso, los punteros para los padres son actualizados, tal que cuando se elija un valor para el gen apuntado actualmente en el hijo, los punteros de los padres muestren las posibles elecciones a realizar. La figura 4.1 muestra este algoritmo.

```
procedure inicializar punteros_padres
begin
  for puntero_hijo = 1 to Long_Cromosoma
   begin
          actualizar punteros de los padres
          hijo.alelo[puntero_hijo] = elección de los valores indicados por los punteros de los padres
     end
end
```

Figura 4.1. Procedimiento para inicializa los punteros padres

Las dos principales características de este procedimiento son el mecanismo de actualización de los punteros en los padres y la manera en que el valor es elegido entre los genes apuntados. Al cambiar la forma en que se seleccionan los valores

a insertarse en el hijo, se pueden definir tres técnicas de scanning: uniforme, basada en la ocurrencia y basada en el fitness (*uniform scanning, ocurrence based scanning y fitness based scanning* en inglés, respectivamente).

4.2.1.1. UNIFORM SCANNING (U-SCAN)

U-Scan es una extensión natural del crossover uniforme para el número de padres. El crossover uniforme se define así:

- Se recorren los (dos) padres y los (dos) hijos de izquierda a derecha.
- Por cada posición en el hijo 1, se elige aleatoriamente si se hereda desde el padre 1 ó desde el padre 2 (el segundo hijo se crea al revertir las decisiones tomadas).

En U-Scan, sólo un hijo es creado. Cada alelo es elegido por un mecanismo aleatorio uniforme, teniendo cada padre la misma oportunidad de ser elegidos para proveer un valor. A continuación se define la probabilidad de heredar del padre *i*, $P(i)$, y el número esperado de genes a heredar del padre *i*, $E(i)$:

$$P(i) = \frac{1}{número\ de\ padres}, \quad E(i) = P(i) * Long\_cromosoma$$

4.2.1.2. OCURRENCE BASED SCANNING (OB-SCAN)

OB-Scan se basa en la siguiente premisa: el valor cuya ocurrencia se encuentre en la mayoría de los padres (los cuales son seleccionados en función de su fitness) en una determinada posición es probablemente el mejor valor a elegir. Es decir que, el alelo con el más alto número de ocurrencias en la posición apuntada se inserta en el hijo [36]. Si el número de ocurrencias de ninguno de los valores es mayoritario, entonces se hereda el valor apuntado en el primer padre. La figura 4.2 muestra OB-Scan para una representación de bits:

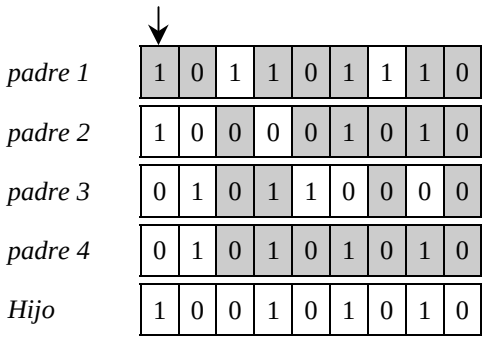

Figura 4.2. OB-Scan sobre patrones de bits

4.2.1.3. FITNESS BASED SCANNING (FB-SCAN)

FB-Scan elige el valor a heredar en forma proporcional al fitness de los padres. Por ejemplo, para un problema de maximización donde el padre *i* tiene un fitness $f(i)$, la probabilidad $P(i)$ de seleccionar un valor desde este padre es (selección proporcional):

$$P(i) = \frac{f(i)}{\sum f(i)}.$$

Al igual que para OB-Scan, el número esperado de genes heredados desde el padre *i* es:

$$E(i) = P(i) * Long\_Cromosoma\ .$$

4.2.1.4. ADAPTACIÓN DEL SCANNING A DIFERENTES TIPOS DE REPRESENTACIÓN

Es posible definir los procedimientos de scanning para diferentes tipos de representación al cambiar el mecanismo de actualización del puntero. Para las representaciones donde todas las posiciones son independientes entre sí (no existe epistasis), el mecanismo de actualización del puntero es simple. Los punteros de los padres son todos iniciados en la primer posición de cada uno de ellos, y en cada caso todos los punteros se incrementan en uno (atravesando a los padres de

izquierda a derecha). Mientras que en aquellos problemas donde existe epistasis, es necesario un mecanismo de actualización de punteros más sofisticado.

La representación basada en el orden [35], necesita un mecanismo de actualización de punteros que asegure que ningún valor se agrega a un hijo dos veces. Por cada padre se incrementa su puntero hasta que este denote un valor que no se haya agregado en el hijo. Un ejemplo de cómo este mecanismo de actualización trabaja se muestra en la figura 4.3:

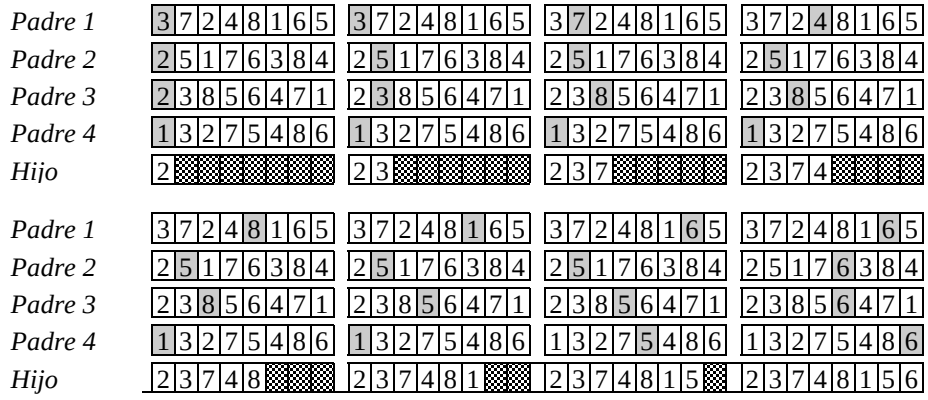

Figura 4.3. OB-Scan sobre una representación basada en el orden.

4.2.2. CROSSOVER BASADO EN LA ADYACENCIA (ABC)

ABC es un caso especializado de scanning [35], específicamente diseñado para representaciones basadas en el orden, donde la posición relativa de los valores es importante, por ejemplo en el TSP. Las principales diferencias entre los procedimientos descriptos en la sección anterior y ABC están en la forma de selección del primer valor y en el mecanismo de actualización de los punteros.

El valor del primer gen del hijo es siempre heredado desde el valor que toma el primer gen en el padre 1. El mecanismo de actualización de los punteros es el siguiente: el puntero de cada padre se mueve al primer sucesor del valor previamente seleccionado que, hasta ese momento, no se encuentre en el hijo.

Un crossover similar a ABC fue propuesto por Whitley [148]. La principal diferencia entre el crossover de Whitley y el ABC ocurre cuando todos los sucesores inmediatos a una ciudad ya han sido heredados al hijo. Por ejemplo, se supone que los posibles sucesores de la ciudad D son las ciudades A, E, F y H (en

los padres 1, 2, 3 y 4 respectivamente), las cuales ya se han incorporado al hijo. El crossover de Whitley elige aleatoriamente una de las ciudades que, todavía, no se han incluido en el hijo. Mientras que, ABC chequea los sucesores de la ciudad A en el padre 1, de la ciudad E en el padre 2, de la F en el 3 y de la H en el 4 (y si cualquiera de estos ya ha sido incluido en el hijo, se buscan sus sucesores), la ciudad a ser agregada en el hijo es elegida entre estos sucesores.

Se pueden definir dos tipos de ABC:

- basado en la ocurrencia (OB-ABC), y
- basado en el fitness (FB-ABC).

Estos crossovers usan el mismo mecanismo para elegir los valores a heredar por los hijos que OB-Scan y FB-Scan. La diferencia está en el mecanismo de actualización del marcador. En la figura 4.4 se muestra un ejemplo de OB-ABC.

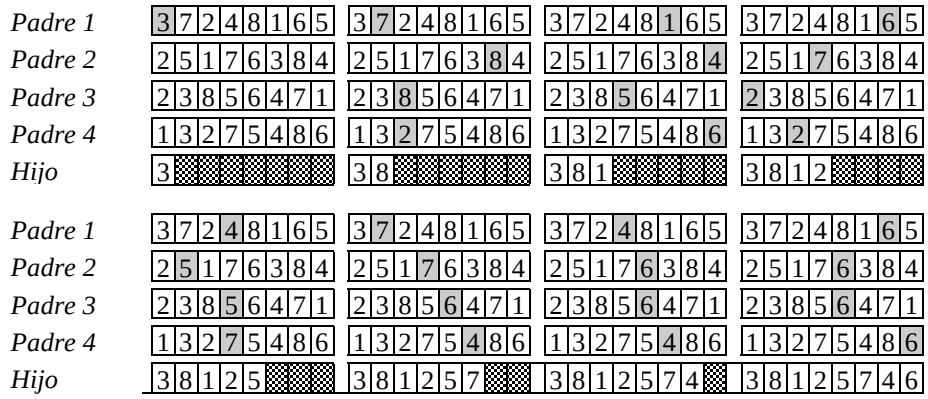

Figura 4.4. OB-ABC.

4.2.3. CROSSOVER DIAGONAL

La idea básica detrás del crossover diagonal es [36, 38]:

- generalizar el mecanismo del crossover de un punto para *n* padres al seleccionar (*n* - 1) puntos de crossover, y
- construir *n* hijos al tomar los *n* segmentos resultantes de los cromosomas padres a lo largo de las diagonales.

En la figura 4.5 se presenta un ejemplo para n=3.

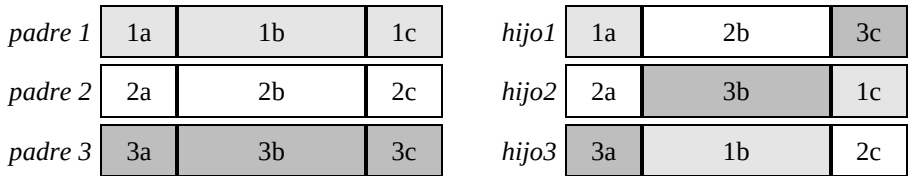

Figura 4.5. Crossover diagonal con tres padres y tres hijos.

4.3. ALGORITMOS EVOLUTIVOS CON MULTIPLES CROSSOVERS

El operador de crossover genera nuevas soluciones al combinar las propiedades de los cromosomas padres. La utilidad de este operador ha sido extensamente discutida por muchos investigadores (por ej. [41, 54, 123, 134]. También se han propuesto una variedad de operadores de crossovers que recorren el espacio de solución en diferentes maneras.

La alternativa de crossover convencional, independientemente del método usado, involucra la aplicación del operador sólo una vez sobre el par de padres seleccionados para crear, a lo sumo, dos hijos [27, 28, 61, 137]; denominado [44] en inglés, *Single Crossover Per Couple* (SCPC). Varias alternativas se han concebido con respecto al número de crossovers que se pueden aplicar por pareja. Estas se describen en las siguientes subsecciones.

4.3.1. MULTIPLE CROSSOVER PER COUPLE (MCPC)

Inspirados en la naturaleza del mundo real Esquivel, Michalewicz, y Gallard [43] introducen la aplicación de más de una operación de crossovers por cada par de padres (pareja), lo que se denomina en inglés, *Multiple Crossover Per Couple* (MCPC). En esta opción el número de hijos puede variar desde uno hasta algún número máximo predefinido.

El número de hijos por pareja se limita a un número máximo y los procesos de producir hijos se controla, por cada par de padres; y de esta manera no exceder el tamaño de la población.

Esta opción permite una economía en el esfuerzo computacional y una mayor explotación de la recombinación de buenas soluciones elegidas previamente. Los distintos experimentos realizados bajo MCPC en [44] muestran como una

desventaja el incremento de la presión selectiva, apareciendo el riesgo de convergencia prematura hacia un óptimo local. Para poder mantener las ventajas de este método sin correr el riesgo antes mencionado, los autores de MCPC introducen un método de selección de la pareja proporcial a su fitness [45] (en inglés *Fitness Proportional Couple Selection*, FPCS). Este método se desarrolla en la sección 4.3.2.

4.3.2. FITNESS PROPORTIONAL COUPLE SELECTION (FPCS)

FPCS divide el proceso de selección en dos pasos; el primero a nivel individual y el segundo a nivel de pareja. El método construye una población intermedia de padres, que se seleccionan individualmente desde la vieja población a través de una selección proporcional, luego se escogen las parejas de acuerdo a su fitness.

Un criterio para asignar el fitness a una pareja está basado en sus disimilitudes. El método puede diseñarse de la siguiente manera [45]:

- Inicialmente se elige por medio de una selección proporcional un número de individuos, para construir la población de padres.
- Luego se evalúa el fitness de una pareja, como el valor absoluto de la diferencia de fitness entre sus componentes.
- Por último se escogen las parejas para la reproducción a través de la selección proporcional. El proceso de la producción de hijos lo controla cada pareja, y de esta manera no se supera el tamaño de la población.

En la figura 4.6 se muestra un esquema general de FPCS. A modo de ejemplo, se supone el siguiente escenario; la pareja *j-ésima* está formada por dos individuos de alto fitness comparable y la pareja *i-ésima* esta compuesta por un individuo de fitness medio y por otro de bajo fitness. Por lo tanto la última pareja tiene mayor oportunidad de ser seleccionada. Este criterio intenta mantener la diversidad genética para evitar el estancamiento pero asume un grado de convergencia más bajo.

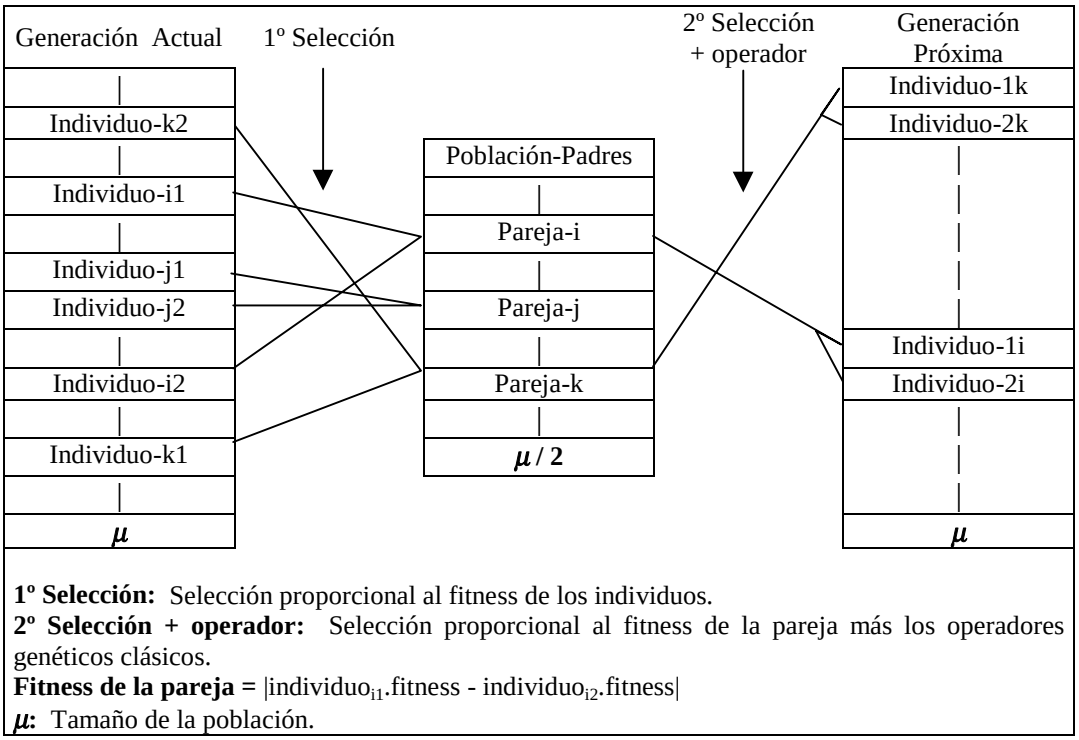

Figura 4.6. Esquema del proceso de selección de parejas

La combinación de MCPC con un método de selección alternativo, como es el FPCS, logra mitigar el problema que ocasiona la presión selectiva cuando se aplica MCPC. Esta conjunción mejora los resultados obtenidos por la aplicación de MCPC únicamente.

4.3.3. AUTO ADAPTACIÓN DE PARÁMETROS PARA MCPC

La auto adaptación es un nuevo campo en la computación evolutiva que permite una actualización dinámica de los parámetros del algoritmo. Evolucionándolo como parte de la estructura del cromosoma. Trabajos previos de Spears [135] sugieren opciones adaptantes para seleccionar el tipo de operador de crossover a aplicarse en cada pareja durante la ejecución del algoritmo evolutivo. Esquivel, Leiva y Gallard en [46] proponen una opción auto adaptante para determinar el número de crossovers a aplicarse a una pareja seleccionada bajo MCPC.

Michalewicz y otros en [3] distinguen tres categorías importantes de control de parámetros:

- *Control de Parámetros Determínistico*: es el caso en que el valor del parámetro se modifica de acuerdo a una regla determinística, sin ningún tipo de retroalimentación del proceso de búsqueda realizado por la estrategia.
- *Control de Parámetros Adaptantes:* en este caso existe alguna información de retroalimentación desde el proceso de búsqueda, que sirve para determinar la dirección y la magnitud del cambio en los parámetros.
- *Control de Parámetros Auto Adaptantes:* aquí se codifican los parámetros a adaptarse son codificados dentro del cromosoma y luego sufren las operaciones genéticas. Los mejores individuos de la población tienen mayores posibilidades de sobrevivir y de reproducirse. Por lo que es de esperar que los mejores valores de parámetros sean propagados con más frecuencia.

Como el número de crossovers a aplicarse en una pareja bajo MCPC, es uno de los parámetros del algoritmo incluido como parte de un individuo; en [46] se presenta una forma de auto adaptación de este parámetro. Por la cual el número de crossovers permitido, se codifica como parte del cromosoma (usando una representación binaria) en las posiciones del extremo derecho del bit string. El conjunto de estas posiciones se denominan *ncross_field*. En general, utilizan los últimos $\log_2(max\_cross+1)$ bits de cada individuo para encontrar el número óptimo esperado de crossovers.

Así, se tiene dos espacios de búsqueda: uno correspondiente a la función objetivo y otro asociado al número de crossovers a aplicar.

En [46] se intenta preservar la información sobre el número de crossovers originalmente aplicado a sus padres. Porque de esta manera y basados en el principio “sobrevive el más fuerte”, las buenas soluciones poseen información sobre el número de crossovers aplicados a sus ancestros, que debería ser apropiado para ellas.

Una vez que la pareja resulta seleccionada se chequea el número correspondiente de crossovers en cada padre, y luego se aplica una técnica local de adaptación[135]:

- Si la cantidad de crossovers a realizar en ambos padres coincide, se aplica este operador el número de veces especificado en *ncross_field*.
- De lo contrario se elige un número aleatorio dentro de un rango permitido.

Pero cuando ocurre el segundo caso, es decir cuando los números decodificados de crossovers son diferentes, se esta violando el intento de preservar información ya que los hijos no mantienen el número de crossovers de sus padres. Si el punto de crossover no divide el *ncross_field*, entonces el hijo retiene información de sus padres, de lo contrario ellos no preservan información sobre cómo fueron creados.

Para retener esa información se han propuesto, en [46], dos opciones:

1. En cualquier situación, el intercambio de información de los padres en los hijos, tradicionalmente, se realiza al aplicar los operadores genéticos con sus correspondientes probabilidades. Los cromosomas padres se eligen y sufren el crossover un cierto número de veces de acuerdo a lo especificado en *ncross_field* si ellos coinciden, o a un valor aleatorio permitido de lo contrario. Después de la recombinación, se aplica la mutación al hijo.
2. En el caso de que no coincidan, esta opción preserva la información de los padres, forzando la diversidad de la población en el espacio de búsqueda de parámetros, ya que la mayor parte de las veces un hijo hereda características de uno de sus padres y el otro hijo las hereda desde el otro padre.

Si los valores especificados en *ncross_field* no coinciden entonces el nuevo valor aleatorio para el número de crossovers se inserta primero en el *ncross_field* del padre, y después se realiza el crossover la cantidad de veces que indique este valor. Esta opción al preservar información individual crea más individuos similares en el espacio de búsqueda paramétrico, incrementando la pérdida de diversidad genética.

4.4. ALGORITMOS EVOLUTIVOS CON MULTIPLES CROSSOVERS SOBRE MÚLTIPLES PADRES

Como consecuencia de las investigaciones relacionadas con la solución de problemas de optimización de múltiples criterios [48], Esquivel y otros [47] han extendido MCPC a múltiples crossovers sobre múltiples padres (en inglés, *multiple crossover per mating* – MCPM), pudiendo obtener más de dos hijos a partir de más de dos padres. Esto se basa en que la opción de múltiples padres de Eiben [35, 36, 37, 38], mitiga la posible pérdida de diversidad generada por MCPC y no necesita los ajustes extras (FPCS) usados anteriormente. Consecuentemente la explotación y la exploración del espacio de búsqueda del problema pueden balancearse.

Como una extensión de MCPC, el multiple crossover per mating (MCPM) provee un medio para explotar las buenas características de los padres seleccionados de acuerdo a su fitness. Una vez seleccionados, los padres sufren la operación de crossover un número n_1 de veces especificadas como argumento, y genera n_2 hijos. Los operadores de crossovers usados son los propuestos por Eiben, descriptos en la sección 2.

CAPÍTULO 5: HEURÍSTICAS CONVENCIONALES PARA TSP

5.1. INTRODUCCIÓN

Las propuestas para resolver el problema del viajante de comercio son muchas. Algunas de ellas están basadas en la programación dinámica o en los métodos de *branch and bound*, los cuales proveen la solución óptima global (después de 4 años con más de 7000 ciudades). Otras opciones más rápidas son los *métodos aproximativos*; pero estas no aseguran el hallazgo de la solución óptima [88]. Algunas de ellas pueden ser: los algoritmos basados en operadores de cambio, el algoritmo de *Lin-Kerninghan*, los algoritmos basados en principios voraces (greedy), simulated anneling, *tabu search*, redes neuronales, los algoritmos evolutivos; pero se han aplicado a un número de ciudades menor a 1000 [139]. En las siguientes secciones se describen algunos de los métodos antes mencionados para resolver el problema del viajante. Los algoritmos evolutivos basados en los genéticos, para TSP son descriptos en el próximo capítulo. Los AEs presentan una excelente performance para resolver problemas de optimización, en especial el de scheduling y el de TSP.

5.2. MÉTODOS HEURÍSTICOS DE BÚSQUEDA PARA RESOLVER EL TSP

5.2.1. BRANCH AND BOUND

Branch and Bound es un método de búsqueda general [132]. Usado para optimizar una función $f(x)$, donde x se encuentra restringido a alguna región factible (definida, por ejemplo, por restricciones matemáticas específicas). Para aplicar branch and bound, es necesario desarrollar etapas similares consecutivas, mediante las cuales se llega a detectar la solución buscada en un proceso ordenado de exploración. Cada Etapa esta constituida por dos fases sucesivas que se cumplen en el orden establecido a continuación [102]:

1. Primera Fase: *Branch* (ramificación o separación).

   Dado un problema, se denomina S_1 al conjunto de soluciones s_k. La primera fase consiste en establecer un criterio de partición de ese conjunto

en base a una propiedad dada. En tal forma se obtienen por lo menos dos subconjuntos S_{11} y S_{12} tales que:

$$S_{11} \cup S_{12} = S_1$$
$$S_{11} \cap S_{12} = \varnothing$$

Esto significa que entre ambos subconjuntos totalizan las soluciones del problema original y que los mismos no tienen elementos en común. Si s_k.es una posible solución del problema, resulta que $s_k \in S_1$ y además se cumple que $s_k \in S_{11}$ ó $s_k \in S_{12}$.

El número de soluciones s_k contenidas en S_{11} y S_{12} puede ser distinto para diferentes criterios de partición. Uno de los subconjuntos obtenidos puede ser vacío o contener una o varias soluciones.

En síntesis, la primer fase consiste en efectuar una partición de un conjunto de acuerdo a una propiedad establecida denominada criterio de partición.

2. Segunda Fase: *Bound* (acotación o evaluación y selección).

   Dada, a partir de la formulación del problema, la función objetivo $f(s)$ a optimizar se supone que se desea obtener el mínimo, entonces es posible encontrar un valor de c_{11} cómo cota inferior de la función para todos los $s_{k1} \in S_{11}$; es decir que:

$$c_{11} < f(s_{k1}).$$

   Esto significa que aún cuando no se calcule el $f(s_{k1})$ para todos los $s_{k1} \in S_{11}$ está identificado un valor que los acota inferiormente a todos ellos. Análogamente resulta para todos los $s_{k2} \in S_{12}$, que:

$$c_{12} < f(s_{k2,}).$$

   En consecuencia, si se comparan c_{11} y c_{12}, resulta más favorable para el caso de mínimo, el subconjunto que posea menor cota. Por ejemplo, si

$c_{12} < c_{11}$, es S_{12} el conjunto seleccionado para cumplir con el nuevo proceso de separación.

En resumen, la segunda fase, al establecer la cota inferior de los valores de $f(s)$ para todas las soluciones de un subconjunto cualquiera, permite también realizar la selección de aquel subconjunto que, no habiendo sido particionado, posea la cota más favorable.

Cabe señalar que cuando el subconjunto está compuesto por un solo elemento la cota elegida debe coincidir con el valor que toma la función para dicha solución.

Cumplidas las fases 1 y 2 se completa una etapa. Si se supone que S_{12} es el subconjunto seleccionado y que el mismo contiene más de un elemento, entonces S_{12} pasa a ser la base de una segunda etapa denominándose S_2 e iniciándose nuevamente las fases 1 y 2 respectivamente. El proceso finaliza cuando el subconjunto, que se seleccione por tener la cota mínima entre todos los subconjuntos no particionados posea un único elemento. El mismo define la solución óptima y su cota, coincidente con el valor que toma la función para dicha solución, representa el mínimo de la función en cuestión.

5.2.2. SIMULATED ANNEALING

Simulated annealing es una opción más compleja para la optimización combinatorial, sugerido inicialmente por Metropolis [104]. Esta basado en observaciones del proceso de enfriado de metales. El término annealing se relaciona con la manera en que los metales en estado líquido son enfriados lentamente para asegurar un estado de energía mínima. Se puede decir que el algoritmo de simulated annealing enfría la solución lentamente hasta alcanzar el objetivo más bajo posible [88]. Esto es, en cada una de las evaluaciones de una solución perteneciente al vecindario, siempre toma un movimiento que mejora la solución.

Sin embargo, permite hacer movimientos que incrementen el costo de la solución basado en una función de probabilidad. Esto supera una de las desventajas asociadas a la búsqueda local. En otras palabras, la facultad de realizar movimientos hacia arriba permite escapar de un mínimo local; explorando

así en forma extensiva el espacio de búsqueda. En los pasos descendientes no existe un mecanismo real para salir de un mínimo local. Algunas técnicas para evitar quedar atrapados en estos puntos, incluyen el incremento del tamaño del vecindario y el uso del muestreo aleatorio pero esto no siempre es completamente satisfactorio.

La función de probabilidad depende del cambio en el costo de un movimiento candidato y de la temperatura actual del sistema. La temperatura es análoga a la temperatura en el proceso de enfriado físico. Es decir, la temperatura se reduce a través del proceso de enfriado. A altas temperaturas cualquier movimiento es permitido, lo que posibilita una extensa exploración del espacio de soluciones; mientras que con las bajas se aceptan pocos movimientos hacia arriba. La idea detrás de la reducción de la temperatura, es que más tarde sobre el proceso de enfriado simplemente se explote el vecindario de una solución óptima local. La temperatura se reduce a través de una planificación adecuada de enfriamiento.

Van Laarhoven y Aarts [142] y más recientemente Reeves [115] concluyen que la performance del algoritmo es fuertemente dependiente de la planificación de enfriamiento empleada.

Las comparaciones entre simulated annealing y los métodos tradicionales para problemas de optimización combinatorial han mostrado resultados de diversa calidad. En una de las investigaciones más cabales Johnson [84, 85] muestra que los algoritmos de simulated annealing obtienen resultados superiores a los tradicionales para el problema de coloreo de grafos, mejores sólo en algunos casos para el problema de particionamiento de grafos.

Simulated annealing se presenta como una atractiva técnica de optimización, la cual es aplicable a una amplia variedad de problemas y relativamente simple de implementar.

La figura 5.2 presenta el algoritmo correspondiente al procedimiento de simulated annealing.

```
procedure Simulated Annealing
begin
      n ← 0;
      inicializar temperatura t;
      seleccionar aleatoriamente un string actual T;
      evaluar T;
      repeat
            repeat
              Seleccionar un nuevo string T' en el vecindario de T al mutar un
              único bit de T;
              if f (T) < f (T')
                    then T ← T';
                    else if  random[0,1) < exp{(  f (T') - f (T))/t}
                          then T ← T';
       until (condición de terminación);
        t ← g(t,n);
       n ← n +1;
     until (criterio de detención);
end
```

Figura 5.1. Procedimiento Simulated Anneling

Donde, la condición de terminación chequea si el equilibrio térmico es alcanzado, en algunas implementaciones este bucle se ejecuta un número *m* de veces. La temperatura *t* es reducida gradualmente a través de la función $g(t,n)$, y en algunos casos *t* es reducido cada cierto número de iteraciones *n*. El algoritmo finaliza con un valor pequeño de *t*, el criterio de detención controla si el sistema está congelado de ser así por ejemplo no aceptará más cambios.

Simulated annealing se ha aplicado con éxito en el TSP [19, 84, 87, 89]. Para TSP simulated annealing es significativamente más lento que Lin-Kernighan, pero este tiene la ventaja que se puede ejecutar por largo tiempo y lentamente mejorar la calidad de los resultados [90]. Obteniendo eventualmente resultados comparables o mejores a los brindados por Lin y Kernighan [87].

5.2.3. TABU SEARCH

El concepto básico de Tabu Search es descripto por Glover en 1986 [65] como una meta-heurística impuesta sobre otra heurística. El objetivo es evitar ciclos al prohibir o penalizar movimientos de la solución, a puntos en el espacio de solución previamente visitados (“tabu”). El método Tabu Search ha sido parcialmente motivado por la observación del comportamiento humano con un

elemento aleatorio que conduce a un comportamiento inconsistente en circunstancias similares. Como Glover señala, la tendencia resultante para desviarse desde de un curso programado, debe ser visto como una fuente de error pero también puede ser probada como fuente de beneficios. Tabu Search procede de acuerdo a la suposición de que no existe una restricción para aceptar una nueva (pobre) solución, a menos que la restricción evite un camino ya investigado. Esto asegura que nuevas regiones del espacio de solución del problema serán investigadas con el objetivo de evitar mínimos locales y encontrar la solución deseada.

Este procedimiento guía a una heurística de escalamiento descendiente con el objetivo de continuar la exploración evitando un retroceso a un óptimo local del cual ya se ha salido. En cada iteración se aplica un movimiento admisible a la solución actual, transformándola de ésta manera en una solución vecina con menor costo. Son permitidas las soluciones que incrementan la función de costo, mientras que el movimiento inverso está prohibido para algunas iteraciones con el objetivo de evitar la ocurrencia de ciclos.

Durante el proceso de búsqueda los movimientos son almacenados en una *lista Tabu*, representando la memoria de los pasos previos del algoritmo. Se cuenta con un proceso para determinar cuando las restricciones Tabu pueden ser sobreescritas.

En muchos casos, las diferencias entre las distintas implementaciones del método Tabu están relacionadas con el tamaño, variabilidad y adaptabilidad de la memoria Tabu a un dominio particular.

La figura 5.3 presenta el algoritmo del procedimiento Tabu Search.

```
procedure Tabu-Searh
begin
      x ← solución inicial factible;
      inicializar nmax con número máximo de iteraciones;
      mejor solución ← x ;
      inicializar el número de iteraciones n = 0;
      inicializar la lista Tabu;
     repeat
       elegir un movimiento no tabu factible Δx (n+1);
       x(n+1) ← x(n) + Δx (n+1);
       if (fobj(x(n+1)) > fobj(mejor solución))
            then   mejor solución ← x(n+1);
       actualizar lista Tabu;
       n ← n+1;
    until (n=nmax or cualquier movimiento posible de la solución actual es Tabu);
end
```

Figura 5.2. Procedimiento Tabu-Search

El dominio de aplicaciones de Tabu Search, tradicionalmente, está conformado por los problemas de optimización combinatorial. La técnica se aplica sencillamente a funciones continuas al elegir una codificación discreta del problema. Muchas de las aplicaciones en la literatura involucran: los problemas de programación entera, scheduling, ruteo, TSP y problemas relacionados.

Dada la flexibilidad inherente al método Tabu Search, no sorprende que una amplia variedad de algoritmos Tabu Search se propusieran para el TSP, por ejemplo, ver [50, 65, 66, 67, 79, 91, 92, 101, 120, 141]. No está claro cuál de estos mecanismos es preferible. Todo ellos requieren un tiempo de ejecución $\Omega(N^3)$ y es poco probable que sean prácticos para instancias de gran tamaño.

5.2.4. REDES NEURONALES

Se define a las estructuras de *redes neuronales* como colecciones de procesadores paralelos conectados entre sí en la forma de grafo dirigido, organizado de tal modo que la estructura de la red sea la adecuada para el problema que se este considerando. Es posible representar esquemáticamente cada elemento de procesamiento de la red como un nodo, indicando las conexiones entre nodos mediante arcos.

Una ventaja muy importante de la aproximación de redes neuronales para resolver problemas, consiste en que no es necesario tener un proceso bien definido

para transformar algorítmicamente una entrada en una salida. Mas bien, lo que se necesita para la mayoría de las redes es una colección de ejemplos representativos de la traducción deseada. La red neuronal se adapta, al entrenarse, para reproducir las salidas deseadas cuando se le presentan las entradas dadas como ejemplo.

Además una red neuronal es robusta, en el sentido de que responderá con alguna salida; incluso en el caso de que se presente una entrada que no haya visto nunca, así como entradas que contengan ruido.

El proceso de entrenamiento es una codificación de la información acerca del problema que hay que resolver, y que la red invierte la mayor parte de su existencia productiva siendo ejercitada una vez que concluye el entrenamiento; entonces se descubre un método para permitir que los sistemas automatizados evolucionen sin ser reprogramados explícitamente.

Cuando se describen las bases matemáticas de los modelos de redes, suele resultar útil pensar que la red es un sistema dinámico; esto es, un sistema que evoluciona a lo largo del tiempo.

También resulta útil ver la colección de valores de los pesos comunes a un sistema dinámico (el aprendizaje es un resultado de la modificación de la fuerza de las uniones sinópticas entre neuronas). El *proceso de aprendizaje* consiste en hallar los pesos que codifican el conocimiento que se desea que el sistema aprenda. Para la mayor parte de los sistemas reales, no es fácil determinar una solución en forma cerrada para este sistema de ecuaciones.

La primera aplicación de una red neuronal para el TSP ha sido la dada por Hopfield and Tank en 1985 [83]. La cual está basada en una formulación de la programación entera del TSP [86].

El algoritmo de Hopfield y Tank regularmente encuentra tours óptimos para instancias de 10 ciudades, aunque a menudo no converge en soluciones factibles cuando el número de ciudades $N=30$, en general no brinda resultados promisorios. Además esta opción es muy sensitiva a los pesos de conexión.

Otra forma de representación de una red neuronal para TSP es la geométrica, donde las neuronas pueden ser vistas como un conjunto de $M \geq N$ puntos en el plano, inicialmente ubicados como los vértices de un polígono regular, de M lados, en el medio de la instancia. El objetivo es mover iterativamente estos

vértices hacia las ciudades, esto significa deformar el polígono. Este proceso continúa hasta que el polígono luzca como un tour, y cada ciudad está representada por un vértice, los vértices que no representen a ninguna ciudad están situados en una línea recta entre otros dos, que sí representan ciudades (figura 5.4). Existen dos variantes en este tema, la red elástica de Durbin y Willshaw [33] y *self-organizing map* derivado de las ideas de Kohonen [93].

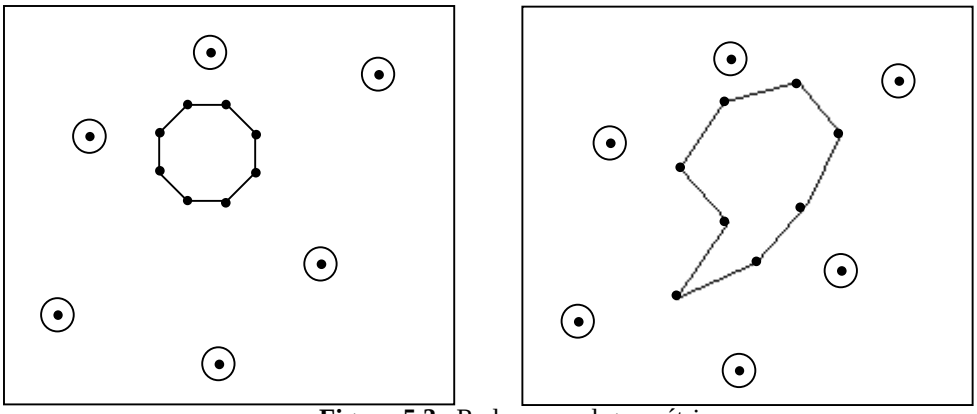

Figura 5.3. Red neuronal geométrica

La primer variante se ha comparado, por sus creadores, con un algoritmo de simulated annealing de calidad desconocida. Siendo los resultados obtenidos para instancias de 50 ciudades un 3% peor que los de ese algoritmo de Simulated Annealing, mientras que para más de 100 ciudades se han observado mejores resultados.

Self-organizing map es una variante de la red elástica, que se ha inspirado en las redes neuronales competitivas de Kohonen [93]. En general los experimentos que se han realizado sobre esta variante concluyen que puede converger más rápidamente que la variante anterior, lo que tiende a producir tours levemente peores [4, 17].

5.2.5. ALGORITMO LIN-KERNIGHAN

Esta es una variante de la búsqueda local en profundidad [103]. Comienza con una noción de la estructura del vecindario de un conjunto de todas las soluciones factibles (tours). Se define el vecindario de un tour, *T* obteniendo los tours

correspondientes al intercambiar a lo sumo k arcos de T. Se inicia con un tour aleatorio T_1 y se construye una secuencia de tours $T_1, T_2,... T_n$. En otras palabras, cada tour es obtenido [99] desde uno previo al realizar *k-cambios*, por ejemplo, al eliminar k enlaces y al reconectar las partes finales desprendidas para lograr un tour. Los *k-cambios* son necesarios para decrementar la longitud del tour. Cuando el proceso se detiene en un tour para el cual no hay posibles mejoras después de *k-cambios*, el tour es *k*-opt. Lin ha introducido y estudiado el caso de $k = 2$ y $k = 3$ y ha mostrado que se pueden obtener tours bastante buenos, rápidamente. Para encontrar un tour óptimo globalmente sugiere repetir la búsqueda desde varios puntos de inicio aleatorios. Posteriormente Lin y Kernighan [100] han realizado una mejora a este algoritmo, la cual consiste en una búsqueda local en ancho (3-Opt) seguida de una búsqueda local en profundidad.

CAPÍTULO 6: ALGORITMOS EVOLUTIVOS AVANZADOS PARA TSP

6.1. INTRODUCCIÓN

Los primeros avances para solucionar el TSP, por medio de Algoritmos Evolutivos han sido introducidos por Goldberg y Lingle en [68] y Grefenstette en [72]. En éste área muchos son los esfuerzos realizados para alcanzar las siguientes características [63]:

- Una apropiada representación para codificar un tour.
- Operadores genéticos válidos para mantener los subtours (o subcircuitos) evitando ilegalidad.
- Prevención de la convergencia prematura.

En función a lo anterior se han creado distintos tipos de representaciones, operadores de crossover y mutación. Entre las formas de representación pueden distinguirse las siguientes: *por permutación*, *por claves aleatorias*, *por adyacencia*, *ordinal* [107, 29, 12]. Mientras que los operadores de crossover usados en el TSP son: *PMX*, *OX*, *Crosover basado en la posición*, *Crossover basado en el orden*, *CX*, *Crossover de intercambio de subtuors*, *Crossover heurístico*, [68, 26, 111, 149, 150, 72]. Por último los operadores de mutación utilizados en este problema son: *por inversión*, *por inserción*, *por desplazamiento*, *intercambio recíproco*, *Mutación heurística* [21, 22].

Cabe destacar que el problema de los algoritmos evolutivos basados en operadores de crossover es el siguiente: que requieren mayor tiempo computacional, por lo que resultan algoritmos costosos. Mientras que los algoritmos basados en operadores de mutación no escapan en forma eficiente de los óptimos locales.

En las secciones siguientes se describen cada una de las formas de: representación, crossover y mutación.

6.2. REPRESENTACIÓN DEL CROMOSOMA

Tradicionalmente los cromosomas son cadenas binarias. Esta simple representación no es conveniente para el TSP y otros problemas combinatoriales. Varios esquemas de representación se han propuesto para el TSP; éstos son descriptos en las secciones 6.2.1 a 6.2.4.

6.2.1. REPRESENTACIÓN POR ADYACENCIA

En la representación por adyacencia [107] un tour se conforma como una lista de *n* ciudades. La ciudad *j* está en la posición *i* de la lista si y solo sí el tour se dirige de la ciudad *i* a la ciudad *j*. Por ejemplo, el vector

(2 4 8 3 9 7 1 5 6)

representa el siguiente tour:

1 – 2 – 4 – 3 – 8 – 5 – 9 – 6 – 7

Cada tour sólo tiene una lista de representación por adyacencia, sin embargo algunas listas pueden representar tours ilegales. Por ejemplo, el vector

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

conduce a:

1 – 2 – 4 –1,

un tour parcial con un ciclo prematuro.

La representación por adyacencia no soporta el operador de crossover clásico (crossover de un punto), por lo tanto resulta necesario un algoritmo de reparación. Los operadores de crossover válidos para esta representación son el *crossover de: arcos alternativos*, *subtours chunks* y el heurístico.

6.2.2. REPRESENTACIÓN ORDINAL

La representación ordinal [107] constituye un tour como una lista de *n* ciudades; el *i*-ésimo elemento de la lista es un número en el rango de 1 a $n - i + 1$. La idea detrás de la representación ordinal es como sigue. Dada una lista de ciudades *C*, que sirve como punto de referencia para listas en representaciones ordinales, se asume por ejemplo que tal lista ordenada es canónica:

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

Dado un tour

1 – 2 – 4 – 3 – 8 – 5 – 9 – 6 – 7

se representa como una lista *l* de referencias,

l = (1 1 2 1 4 1 3 1 1),

y debería interpretarse como sigue:

- El primer número sobre la lista *l* es 1, entonces toma la primer ciudad desde la lista *c* como la primer ciudad del tour (ciudad número 1), y la elimina desde *C*. El tour parcial es:

  1 –

- El número próximo sobre la lista *l* es también 1, entonces toma la primer ciudad de lista *C* actualizada con la próxima ciudad del tour (ciudad número dos) y la elimina de *C*. El tour parcial es:

  1 – 2

- El número próximo sobre la lista *l* es 2, entonces toma la segunda ciudad de lista *C* actualizada con la próxima ciudad del tour (ciudad número cuatro) y la elimina de *C*. El tour parcial es:

  1 – 2 – 4

- Así siguiendo con cada uno de los elementos de la lista *l* hasta llegar al final de la misma.

La principal ventaja de la representación ordinal es que el operador de crossover clásico (crossover de un punto) funciona. Un ejemplo de esto se muestra en la figura 6.1.

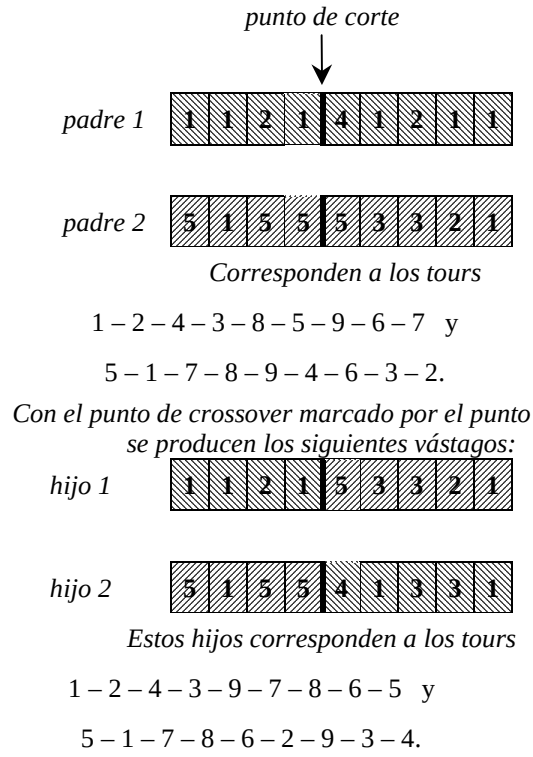

Figura 6.1. Ilustración del operador de crossover clásico bajo una representación ordinal.

Es fácil ver que los tours parciales a la izquierda del punto de crossover no cambian, mientras que los tours parciales a la derecha del punto de crossover son desorganizados de forma bastante aleatoria. Los pobres resultados experimentales

obtenidos [72] indican que esta representación junto con el operador de crossover clásico no es apropiada para el TSP.

6.2.3. REPRESENTACIÓN DEL CAMINO (POR PERMUTACIÓN)

Esta representación es por supuesto la forma más natural de presentar un tour para el TSP, donde las ciudades son listadas en el orden en el cual son visitadas [29, 107]. El espacio de búsqueda para ésta representación es el conjunto de permutaciones de las ciudades. Por ejemplo, un tour para el TSP de 9 ciudades

3 – 2 – 5 – 4 – 7 – 1 – 6 – 9 – 8

es simplemente representado como sigue:

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

Esta representación puede conducir a tours ilegales si se utiliza el crossover de un punto. Muchos operadores de crossover se han investigado por esto. Los más conocidos son: PMX, CX y OX.

6.2.4. REPRESENTACIÓN POR CLAVES ALEATORIAS.

Es la primer representación introducida por Bean [12]. La cual codifica la solución con números aleatorios en el intervalo abierto (0,1). Estos valores se usan como claves de ordenamiento para codificar la solución. Por ejemplo, un cromosoma para un problema de 9 ciudades puede representarse como:

(0.23 0.82 0.45 0.74 0.87 0.11 0.56 0.69 0.78),

donde la posición *i* en la lista representa la ciudad *i*. El número aleatorio en la posición *i* determina el orden de visita de la ciudad *i* en un tour TSP. Para este

ejemplo se ordenan las claves en orden ascendente, obteniéndose así el siguiente tour:

6 – 1 – 3 – 7 – 8 – 4 –9 – 2 – 5

Las claves aleatorias eliminan la posibilidad de hijos no factibles al representar soluciones de la forma antes descripta. Esta representación es aplicable a una gran variedad de problemas de optimización secuenciales incluyendo el scheduling de máquinas, alocación de recursos, ruteo vehicular, problemas de asignación cuadrática.

6.3. OPERADORES DE CROSSOVER

Varios operadores de crossover se han propuesto para el TSP, tales como *Partial-Mapped Crossover* (PMX), *Order Crossover* (OX), *Cycle Crossover* (CX), Crossover Basado en la Posición, Crossover Basado en el Orden, Crossover Heurístico, Crossover de Subtours Chunks, Crossover por Arcos Alternativos (Edge Recombination, *ER*), entre otros. Estos operadores pueden clasificarse en:

- canónicos y
- heurísticos.

Los operadores canónicos es posible verlos como una extensión del crossover de dos puntos al crossover multipunto de tours binarios. Generalmente la representación por permutación produce vástagos ilegales al usar crossover de dos puntos o multipunto, ya que algunas ciudades pueden perderse, mientras que otras pueden duplicarse en el vástago. En estos casos los procedimientos de reparación se usan para resolver la ilegitimidad del hijo.

La esencia de la opción canónica es el mecanismo aleatorio. No hay garantía de que un hijo producido por este tipo de crossover sea mejor que sus padres. La aplicación de heurísticas en el crossover intenta generar un vástago mejorado.

6.3.1. PARTIAL MAPPED-CROSSOVER (PMX)

Goldberg y Lingle presentan PMX en [68]. Es posible ver a este operador como: una extensión del crossover de dos puntos para un tour binario a una representación por permutación. Este usa un procedimiento de reparación especial para resolver la ilegitimidad causada por el simple crossover de dos puntos. Es decir, las bases de PMX son el crossover de dos puntos más un procedimiento de reparación. PMX opera de la siguiente manera:

1. Selecciona, uniformemente al azar, dos posiciones a lo largo del tour. Los *subtours* definidos por las dos posiciones se *denominan secciones de transferencia* (o mapping sections en inglés).
2. Intercambia dos subtours entre los padres para producir 2 hijos.
3. Determina las relaciones de transferencia entre las dos secciones transferidas.
4. Legaliza los hijos con las *relaciones de transferenica*.

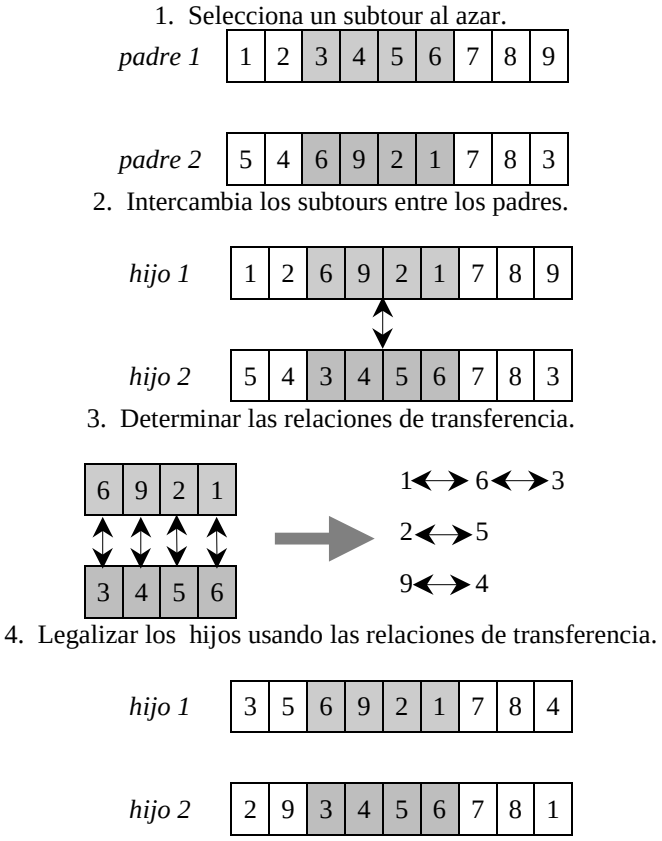

Figura 6.2. Ilustración del operador PMX

La figura 6.2 ilustra el procedimiento del PMX. Las ciudades 1, 2, y 9 están duplicadas en el hijo 1, mientras que las ciudades 3, 4, y 5 son perdidas. De acuerdo a las relaciones de transferencia determinadas en el paso 3, las ciudades repetidas 1,2, y 9 deberían reemplazarse por las ciudades faltantes 3, 5, y 4, respectivamente, mientras mantiene el subtour intercambiado sin modificaciones.

6.3.2. ORDER CROSSOVER (OX)

OX es propuesto por Davis [26]. Este puede verse como una variación de PMX con un procedimiento de reparación diferente. OX trabaja como sigue:

1. Selecciona un subtour desde un padre al azar.
2. Produce un hijo al copiar el subtour dentro sus correspondientes ubicaciones.
3. Borra las ciudades que están en el subtour del segundo padre. La secuencia de ciudades resultantes son las requeridas por los hijos.
4. Ubica las ciudades en las posiciones libres del hijo de izquierda a derecha, de acuerdo al orden de la secuencia para producir un vástago.

La figura 6.3 ilustra este procedimiento. Esta muestra un ejemplo de cómo se obtiene un cromosoma hijo. Con idénticos pasos, podemos producir el segundo hijo [2 5 4 9 1 3 6 7 8] desde los mismos padres.

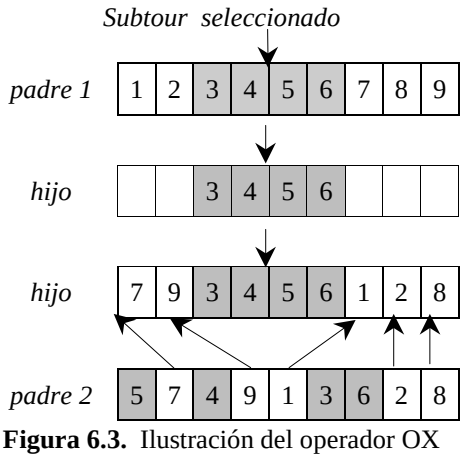

Figura 6.3. Ilustración del operador OX

6.3.3. CROSSOVER BASADO EN LA POSICIÓN

Syswerda presenta el crossover basado en la posición en [26]. Este es esencialmente un tipo de crossover uniforme para una representación por permutación junto con un procedimiento de reparación. Esto puede visualizarse también como una variación de OX en el cual las ciudades se seleccionan en forma no consecutiva. El operador de crossover basado en la posición funciona como sigue:

1. Selecciona un conjunto de posiciones al azar desde un padre.
2. Produce un hijo al copiar las ciudades sobre estas posiciones en las correspondientes ubicaciones del hijo.
3. Borra las ciudades que son seleccionadas desde el segundo padre. La secuencia resultante contiene las ciudades que necesita el hijo.
4. Ubica las ciudades en las posiciones libres del hijo de izquierda a derecha de acuerdo al orden de la secuencia; y así producir un hijo.

El procedimiento anterior se ilustra en la figura 6.4. Con los mismos pasos, se puede producir el segundo hijo como [2 4 3 5 1 9 6 7 8].

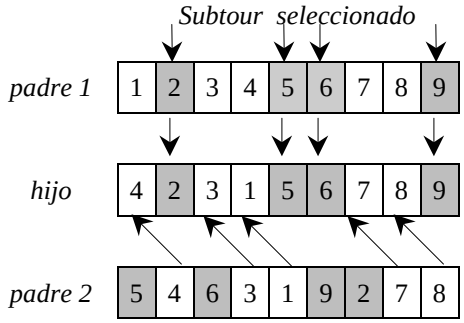

Figura 6.4. Ilustración del operador de crossover basado en la posición.

6.3.4. CROSSOVER BASADO EN EL ORDEN

El operador de crossover basado en el orden lo introduce Syswerda en [26]. Este presenta una leve variación al operador de crossover basado en la posición en la cual el orden de las ciudades seleccionadas en un padre, es impuesta sobre las ciudades correspondientes en el otro padre. Como muestra la figura 6.5. Con los mismos pasos, se puede generar el segundo hijo como [4 2 3 1 5 6 7 9 8].

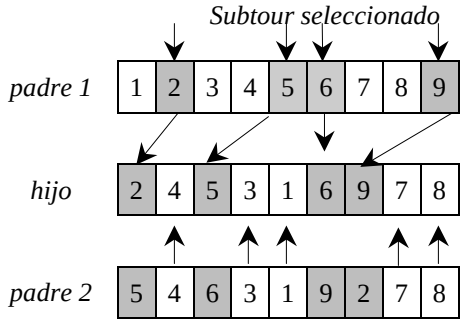

Figura 6.5. Ilustración del operador de crossover basado en la posición.

6.3.5. CYCLE CROSSOVER (CX)

Oliver, Smith y Holland proponen CX en [111]. Al igual que el crossover basado en la posición, este toma las mismas ciudades desde un padre y selecciona las ciudades restantes desde el otro padre. La diferencia radica en que las ciudades del primero no se seleccionan aleatoriamente, sino que se eligen si definen un ciclo de acuerdo a las posiciones correspondientes entre padres. CX funciona de la siguiente [60]forma:

1. Encuentra el ciclo definido por las posiciones correspondientes de las ciudades entre los padres.
2. Copia las ciudades en el ciclo a un hijo con las posiciones correspondientes de un padre.
3. Determina las ciudades restantes para el hijo al eliminar las ciudades que ya están en el ciclo desde el otro padre.
4. Completa al hijo con las ciudades restantes.

Este procedimiento se ilustra en la figura 6.6. Con los mismos pasos se puede crear el segundo vástago.

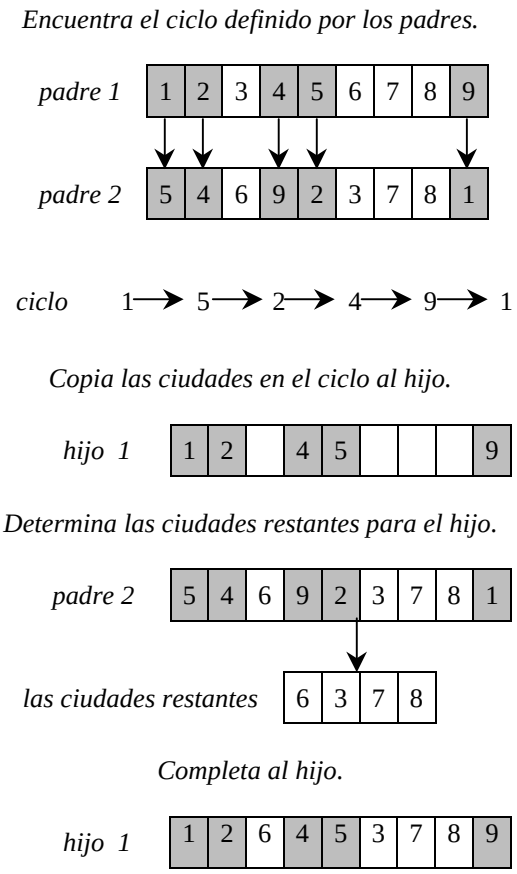

Figura 6.6. Ilustración del operador CX.

6.3.6. SUBTOUR EXCHANGE CROSSOVER

Yamamura, Ono, y Kobayashi proponen este operador de crossover [149, 150]. Primero selecciona subtours desde los padres. Los cuales contienen las ciudades en común. Los hijos son creados al intercambiar los subtours como lo muestra la figura 6.7.

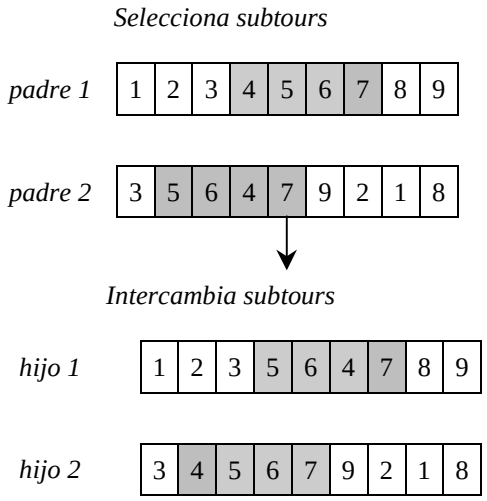

Figura 6.7. Ilustración del operador subtour exchange crossover.

6.3.7. CROSSOVER HEURÍSTICO

El crossover heurístico es propuesto por Grefenstette et. al en [72]. En heurísticas convencionales para el TSP, existen dos alternativas de construcción básicas: heurísticas del *vecino más cercano*, y la de *mejor inserción*. El crossover presentado por Grefenstette se implementa con la heurística del vecino más cercano. Cheng y Gen han diseñado un crossover con el mecanismo de la mejor inserción para el problema del ruteo vehicular y del viajante. El crossover heurístico funciona así:

1. Dado un par de padres se elige una ciudad al azar, para el inicio.
2. Se elige el arco más corto (representado en los padres) que va desde la ciudad actual a aquellas ciudades que no forman un ciclo. Si los dos arcos conducen a un ciclo, se elige, en forma aleatoria, una ciudad que continúe el tour.
3. Si se completa el tour, el procedimiento se detiene; de otra forma se repite el paso 2.

Liepins presenta una leve modificación de la versión de Grefenstette [98], la cual toma cualquier tour siempre que comience en la misma ciudad. Cheng y Gen también han propuesto una versión modificada del crossover heurístico [21]. Esta

presenta un *procedimiento de rotación de subtour* para heredar a los mejores subtours de los padres en los hijos.

6.3.8. CROSSOVER POR ARCOS ALTERNATIVOS

Michalewicz implementa este operador [107] para la representación por adyacencia. Construye un hijo al elegir aleatoriamente un arco desde el primer padre, entonces selecciona el arco apropiado desde el segundo progenitor, etc. El operador extiende el tour al elegir arcos desde los padres alternativos. Si el nuevo arco (desde uno de los padres) introduce un ciclo al tour actual, entonces el operador selecciona uno al azar, que no introduzca un ciclo, desde los arcos restantes.

Por ejemplo, el primer hijo de dos padres puede ser el presentado (*hijo 1*)en la figura 6.8.

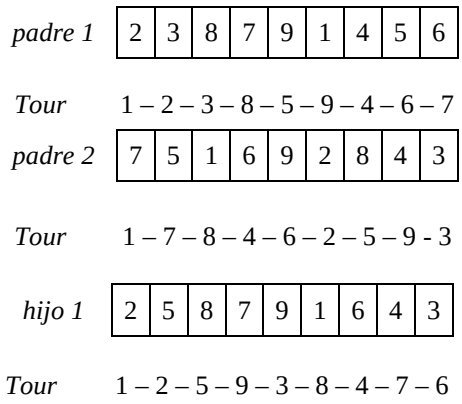

Figura 6.8. Ilustración del operador por arcos alternativos

Donde el proceso se inicia en el arco (1,2) desde el *padre 1* y el único arco introducido aleatoriamente durante el proceso de los arcos alternativos es (7,6) en vez de (7,8). El cual introduciría un ciclo prematuro.

6.3.9. CROSSOVER POR SUBTOURS CHUNKS

El crossover por Subtours Chunks [107], al igual que el presentado en la sección anterior se utiliza con la representación por adyacencia.

Este operador de crossover construye un hijo al elegir un subtour (de longitud aleatoria) desde uno de los padres, y luego se elige otro desde uno de los padres restantes (también de longitud aleatoria). El operador extiende el tour al seleccionar arcos desde padres alternativos. Si algún arco (pertenecientes a alguno de los padres) introduce un ciclo en el tour actual, el operador selecciona, en forma aleatoria, un arco a partir de los arcos restantes que no incluyan ciclos.

6.3.10. CROSSOVER POR RECOMBINACIÓN DE ARCOS

Whitley, Starweather y Fuquay introducen este operador de crossover [146, 107]. ER transfiere más del 95% de los arcos desde los padres a un único hijo y explora la información sobre los arcos en un tour, por ejemplo para el tour

3 – 1 – 2 – 8 – 7 – 4 – 6 – 9 – 5

los arcos son:

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

Luego todos los arcos -no las ciudades- poseen valores (distancias) en el TSP. Entonces la función objetivo, a minimizarse, es el total de arcos que constituyen un tour legal. La posición de una ciudad en un tour no es importante, ya que los tours son circulares. Tampoco lo es la dirección de un arco; porque los arcos (3 1) y (1 3), por ejemplo, indican sólo que las ciudades 1 y 3 están directamente conectadas entre sí.

La idea general detrás del crossover ER es que un hijo debería construirse exclusivamente desde los arcos presentes. Esto se realiza con ayuda de la lista de arcos creada desde los dos tours padres. La lista de arcos provee, por cada ciudad *c*, todas las otras ciudades conectadas a la ciudad *c* en al menos uno de los padres. En todo grafo, por cada ciudad *c* existe al menos dos y a lo sumo 4 ciudades en la lista. Un ejemplo de esto, se observa en la figura 6.9.

padre 1	1	2	3	4	5	6	7	8	9
padre 2	4	1	2	8	7	6	9	3	5

La lista de arcos es:

Ciudad 1: arcos a otras ciudades: 9 2 4
Ciudad 2: arcos a otras ciudades: 1 3 8
Ciudad 3: arcos a otras ciudades: 2 4 9 5
Ciudad 4: arcos a otras ciudades: 3 5 1
Ciudad 5: arcos a otras ciudades: 4 6 3
Ciudad 6: arcos a otras ciudades: 5 7 9
Ciudad 7: arcos a otras ciudades: 6 8
Ciudad 8: arcos a otras ciudades: 7 9 2
Ciudad 9: arcos a otras ciudades: 8 1 6 3

Figura 6.9. Ilustración del operador de crossover ER

La construcción del hijo comienza con la selección de una ciudad de inicio desde uno de los padres. Entonces, se elige la ciudad conectada a la inicial con el menor número de arcos. Si estos números son iguales, se realiza una elección aleatoria entre ellos. Cada selección incrementa la posibilidad de completar un tour con todos los arcos seleccionados desde los padres. Con una selección al azar, la chance de tener un arco erróneo debiera ser mucho más grande.

Se asume que la ciudad 1 es seleccionada. Esta ciudad está directamente conectada con otras tres: 9, 2, y 4. La próxima ciudad se elige desde una de estas tres. En este ejemplo, las ciudades 4 y 2 tienen tres arcos, y la ciudad 9 tiene 4. Se hace una selección aleatoria entre las ciudades 4 y 2; se asume que se ha seleccionado la ciudad 4. Siendo el tour parcial resultante:

1 – 4

Nuevamente los candidatos para la próxima ciudad del tour, que se está construyendo, son 3 y 5, si ellos están directamente conectados a la última ciudad (4), entonces se selecciona la 5. Siendo el tour parcial resultante:

1 – 4 – 5

Continuando con este procedimiento se obtiene el hijo:

1 – 4 – 5 – 6 – 7 – 8 – 2 – 3 – 9

el cual se forma completamente por arcos tomados desde ambos padres.

6.4. OPERADORES DE MUTACIÓN

Es relativamente fácil producir algunos operadores de mutación para representación del camino. Algunos de ellos pueden ser la mutación por: inversión, inserción, desplazamiento e intercambio recíproco [21, 22].

6.4.1. MUTACIÓN POR INVERSIÓN

La mutación por inversión selecciona dos posiciones dentro de un cromosoma al azar y entonces invierte el subtour entre estas dos posiciones, como se ilustra en la figura 6.10.

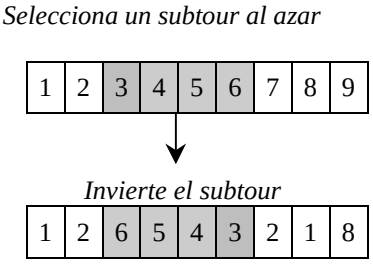

Figura 6.10. Ilustración del operador de mutación por inversión.

6.4.2. MUTACIÓN POR INSERCIÓN

La mutación por inserción selecciona una ciudad al azar y la inserta en una posición aleatoria, como ilustra la figura 6.11.

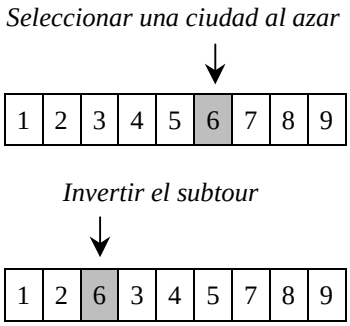

Figura 6.11. Ilustración del operador de mutación por inserción.

6.4.3. MUTACIÓN POR DESPLAZAMIENTO

Este tipo de mutación selecciona un subtour al azar y lo inserta en una posición aleatoria, como se ilustra en la figura 6.12. La inserción puede verse como un caso especial de desplazamiento en el cual el subtour contiene sólo una ciudad.

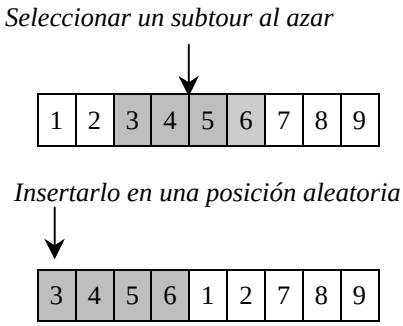

Figura 6.12. Ilustración del operador de mutación por desplazamiento.

6.4.4. MUTACIÓN POR INTERCAMBIO RECÍPROCO

La mutación por intercambio recíproco selecciona dos posiciones aleatorias y luego intercambia las ciudades sobre estas posiciones, como ilustra la figura 6.13. Esta mutación es esencialmente la heurística 2-Opt para TSP.

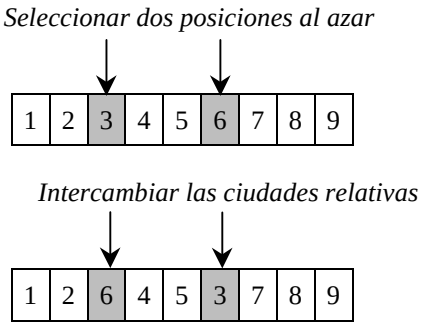

Figura 6.13. Ilustración del operador de mutación por intercambio recíproco.

6.4.5. MUTACIÓN HEURÍSTICA

Cheng y Gen proponen este tipo de mutación [21, 22]. Está diseñada con la técnica del vecindario para producir, así, un hijo mejorado. Un conjunto de cromosomas se transforman a partir de un cromosoma dado al intercambiar no más de λ genes. Este conjunto de cromosomas obtenido es considerado como el vecindario. Entonces, el mejor de estos vecinos es tomado como el hijo producido por la mutación. Este operador sigue los siguientes pasos:

1. Toma λ genes aleatoriamente.
2. Genera vecinos de acuerdo a todas las permutaciones posibles de los genes seleccionados.
3. Evalúa a todos los vecinos y selecciona el mejor como hijo.

La figura 6.14 muestra este procedimiento.

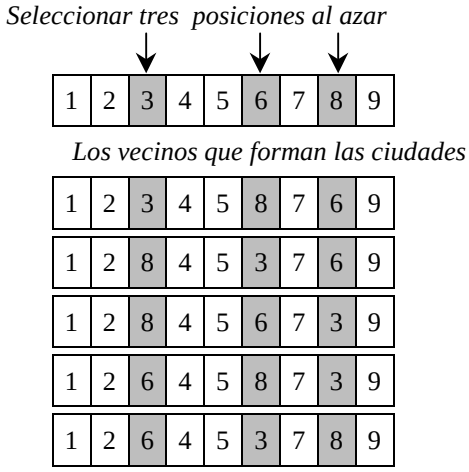

Figura 6.14. Ilustración del operador de mutación heurística

CAPÍTULO 7: UN ALGORITMO EVOLUTIVO AVANZADO PARA TSP

7.1. INTRODUCCIÓN

En función a los problemas presentados por los algoritmos evolutivos para resolver eficiente y eficazmente el problema del viajante de comercio, Michalewicz en [109, 139] propone un algoritmo evolutivo basado en un operador especial denominado *inver-over*. Este operador incorpora, en una solución, el conocimiento tomado desde otros individuos de la población. El cual fusiona la inversión y la recombinación. La inversión se aplica a una parte de un único individuo, pero la selección del segmento a invertir depende de otros individuos pertenecientes a la población. El método provee soluciones óptimas o muy cercanas a las mismas velozmente, superando así a otros operadores evolutivos propuestos en el pasado (ver en el capítulo 5). Se explica este método en la sección 7.2.

Como las opciones multirecombinativas [35, 44, 47], que permiten la múltiple aplicación de crossovers sobre los padres seleccionados, han mostrado una mejor performance en problemas de optimización altamente combinatoriales en [110] se muestra una variante del operador inver-over llamada *multi-inver-over*. La cual permite explotar las características de los contribuyentes intervinientes de una nueva solución en el espacio de búsqueda. En la sección 3 se desarrolla esta variante.

En las secciones 4 y 5 se presenta una comparación de los resultados obtenidos con ambas alternativas.

7.2. DESCRIPCIÓN DEL ALGORITMO EVOLUTIVO CON EL OPERADOR *INVER-OVER*

En el algoritmo evolutivo donde se aplica el operador inver-over, cada individuo compite con su hijo. Aquí se usa únicamente este operador, adaptativo, que toma como guía a la población actual. Mientras que sus componentes adaptativos son: el número de inversiones promedio aplicadas a un único

individuo y el segmento a ser invertido (determinado por otro individuo elegido al azar). Además, también es variable el número de veces que el operador se aplica a un individuo.

De lo anterior es posible deducir las siguientes características del algoritmo evolutivo con el operador inver-over [109]:

- Existe una población P con μ individuos, donde $P = \{S_1, S_2, \ldots\ldots\ldots\ldots, S_\mu\}$.
- La representación cromosómatica usada es la representación por permutación.
- Cada individuo compite sólo con su hijo.
- Se utiliza un solo operador de inversión, que no es aleatorio sino adaptativo. Este operador toma como guía a la población actual.
- El número de veces que el operador se aplica a un individuo durante una única generación, es variable. Los componentes adaptativos son: el número de inversiones promedio aplicadas a un único individuo y el segmento a ser invertido (determinado por otro individuo elegido al azar).

La figura 7.1 detalla el algoritmo que describe el uso del operador adaptativo inver-over. Donde *p* es la probabilidad de que la segunda ciudad sea seleccionada aleatoriamente, *rand( )* es una función que genera un número (perteneciente a los reales positivos) en el rango [0, 1], y *eval(S)* es una función que evalúa al individuo *S* calculando la suma de las distancias euclideanas entre las ciudades que forman el ciclo hamiltoniano.

Este algoritmo puede verse como *m*-procedimientos hill-climbing en paralelo. En este procedimiento, el número de inversiones y el segmento a invertirse dependen de la población actual. Este es un algoritmo evolutivo, con un operador adaptativo, el cual es una combinación de las operaciones de inversión y crossover, esto es, cuando la segunda ciudad es seleccionada en base a otro individuo de la población.

Si *rand( ) > p* un segundo individuo, elegido en forma aleatoria desde la población *P*, provee una guía para el segundo marcador de la inversión. Si el operador de inversión es similar al crossover, parte del patrón del segundo individuo aparece en el hijo.

```
Inicialización aleatoria de la población P
while (la condición la condición de terminación no sea satisfecha) do
        for cada individuo Si ∈ P do{
            S' = Si
        selecciona (aleatoriamente) una ciudad c desde S'
            repeat{
            if (rand( ) ≤ p)
                    selecciona la ciudad c' desde las ciudades restantes en S'
                else{
                    select (randomly) an individual from P
                    asigna a c' la "siguiente" ciudad a la ciudad c en el individuo seleccionado
                }
                if (la ciudad siguiente o la ciudad previa de la ciudad c a la ciudad c' en S'
                  exit desde el loop repeat
            invertir la la sección desde la ciudad siguiente de la ciudad c a la ciudad c' en S'
                c = c'
             }
             if (eval(S') ≤ eval(Si))
                Si = S'
        }
}
```

Figura 7.1. El Algoritmo Evolutivo usando el operador *inver-over*

Para ilustrar esta operación se presenta el siguiente ejemplo. Se supone un total de 8 ciudades, es decir, $m = 8$, la siguiente representación (o individuo) describe un tour:

$$S' = (2,8,7,6,5,4,3,1)$$

y que la actual ciudad, c, es 8. Si $rand( ) \leq p$, otra ciudad, c', del mismo individuo S' es seleccionada al azar; para este ejemplo c' es 3, y el segmento correspondiente es invertido creando el siguiente vástago:

$$S' = (2,8,3,4,5,6,7,1)$$

En cambio si *rand( ) > p* se selecciona otro individuo desde la población, por ejemplo:

I = (6,4,7,2,8,5,1,3)

Entonces en este último individuo, *I*, se busca la ciudad $c´$ siguiente a c = 8, es decir que $c´$ = 5, por lo tanto la sección a invertirse en $S´$ comienza después de la ciudad 8 y termina en la 5. Por ende el hijo generado es el que se muestra a continuación:

$S´$= (2,8,5,6,7,4,3,1).

Por último si $eval(S´) \leq eval(S_i)$ entonces el individuo S_i es reemplazado por $S´$ en la población actual. Es por esto, que el padre compite con su hijo.

7.3. EL OPERADOR *INVER-OVER* Y LA MULTIRECOMBINACIÓN

Las opciones de multirecombinación, presentadas en el capítulo 3, forman parte de una familia más amplia de algoritmos evolutivos. Los cuales incluyen múltiples contribuyentes y operadores para intercambiar material genético y son denominados en inglés como "*Multiplicity Feature Evolutionary Algorithms*" (MFEA).

Con una nueva perspectiva a la naturaleza del operador, MFEAs puede también ser divisado por el operador inver-over [110]. La idea es continuar aplicando sobre el mismo individuo $S´$, un número predeterminado n_1 de veces, esperando encontrar mejores soluciones. Esto es, al comparar las evaluaciones de $S´$ y S_i , si $S´$ no supera a S_i entonces la operación de inver-over es repetida sobre S_i; siendo n_1 la máxima cantidad de veces que esta operación se puede repetir. Esta reiteración permite que otros individuos de la población compitan como donadores hasta que eventualmente un mejor vástago sea creado y reemplace a S_i en la población actual.

7.4. EXPERIMENTACIÓN Y ANÁLISIS DE RESULTADOS

Se han realizado cinco diferentes experimentos aplicando los métodos descriptos en las secciones 7.2 y 7.3, *IO-1* a *IO-5*. Donde *IO-1* representa la aplicación del operador inver-over tradicional. Mientras que *IO-2* a *IO-5* representan al algoritmo de multirecombinación donde el valor numérico indica la cantidad n_1 de veces que fue aplicado el operador. Todos ellos se han probado sobre siete instancias para el TSP extraídas de TSPLIB95 [116]. La tabla 7.1 presenta estas instancias:

IDENTIFICADOR	*INSTANCIA*	*MEJOR VALOR CONOCIDO*
Eil51	Eil51 (51 ciudades)	426
Eil76	Eil76 (76 ciudades)	538
KroA100	KroA100 (100 ciudades)	21282
KroD100	KroD100 (100 ciudades)	21294
Pr76	Pr76 (76 ciudades)	108159
St70	St70 (70 ciudades)	675
Bier127	Bier127 (127 ciudades)	118282

Tabla 7.1. Instancias para el TSP

Para cada instancia se han llevado a cabo una serie de diez ejecuciones. En la tabla 7.2 se describen la configuración de los parámetros que se usaron en todos los algoritmos evolutivos con operador inver-over:

Tamaño de la población:	100, 200
Probabilidad *p*:	0.02
Nº máximo de generaciones:	4000
Elitismo:	Sí

Tabla 7.2. Parámetros

Como criterio de terminación se ha decidido detener al algoritmo si el mejor individuo se mantiene sin cambios después de 50 generaciones consecutivas a partir de la generación 200. Como una indicación de la performance del algoritmo se evalúan las siguientes variables:

- ***Ebest*** = (abs (*opt_val* – mejor valor) / *opt_val*)100. Esto es el error porcentual del mejor individuo encontrado cuando es comparado con el valor conocido, estimado u óptimo *opt_val*. Esto da una medida de cuán alejado se está del *opt_val.*
- ***Epop*** = (abs (*opt_val* – fitness promedios poblacional) / *opt_val*)100. Esto es el error porcentual del fitness promedio poblacional cuando es comparado *opt_val*. Esto da una medida de cuán alejado está el individuo promedio del *opt_val.*
- ***Gbest:*** Identifica la generación donde el mejor individuo (retenido por elitismo) fue encontrado.

La tabla 7.3 muestra resultados detallados para la instancia Eil51 bajo cada opción. Los valores promedios para las variables antes mencionadas se muestran en las columnas 2, 3 y 4. La columna 5 muestra el valor de Ebest correspondiente al mejor individuo encontrado en los experimentos.

Instancia Eil51				
Opción	*Gbest Promedio*	*Ebest Promedio*	*Epop Promedio*	*Ebest Mínimo*
IO-1	531.5	7.034321455	19.82355847	4.739703521
IO-2	411	2.877604296	10.69841458	1.14099061
IO-3	313.1	2.495894155	9.628347113	1.543312676
IO-4	271.2	1.805229531	8.646420681	0.674121127
IO-5	276	1.552542559	7.297843404	0.82870493

Tabla 7.3. Valores de las variables de performance para la instancia Eil51.

Es posible observar que todas las opciones que implementan multirecombinación tienen un comportamiento mejor que el presentado por la opción *IO-1*. Considerando la calidad de los individuos mejores y promedios, los valores medios de *Ebest* y *Epop* son mayores sobre *IO-1* para esta instancia. Esto se repite para el *Ebest* mínimo.

Las siguientes figuras muestran un comportamiento promedio para todas las instancias del TSP bajo cada alternativa evolutiva, aquí presentada (*IO-1* a *IO-5*).

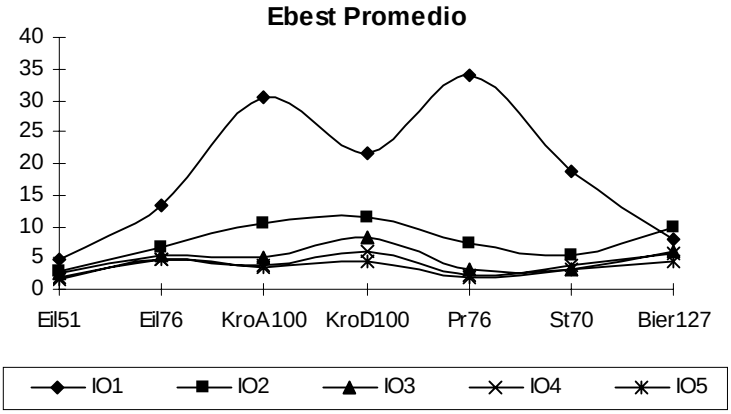

Figura 7.2. Promedio de las medias de los valores de Ebest para todas las instancias bajo cada alternativa.

La figura 7.2, indica la media de los valores de Ebest promedio para todas las instancias bajo cada alternativa. También aquí, se observa un mejor comportamiento promedio bajo las opciones de multirecombinación. Los mejores valores se obtienen con valores de $3 \leq n_1 \leq 5$, destacándose los resultados obtenidos bajo *IO-5*. El cual presenta valores en el rango de 1.6% para Eil51 a 4.7% para Eil76. Los peores resultados se obtienen bajo *IO-1*, en el rango de 7.0% para Eil51 a 34.1% para Pr76.

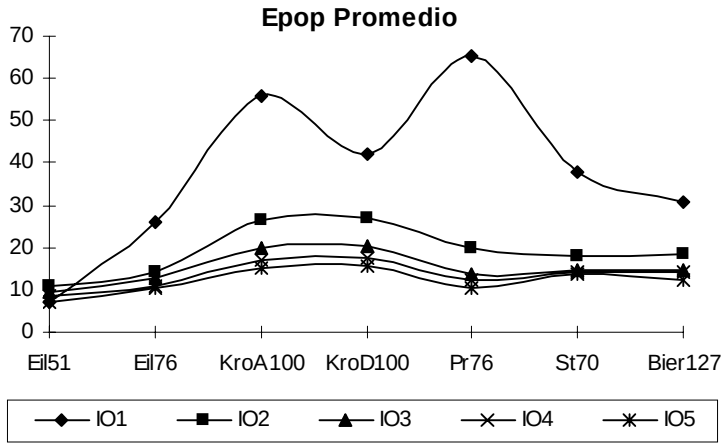

Figura 7.3. Promedio de las medias de los valores de Epop para todas las instancias bajo cada alternativa.

La figura 7.3, muestra la media de los valores de *Epop* promedio para todas las instancias bajo cada opción. En la cual se observa un comportamiento promedio similar. Donde además, los mejores valores se obtienen con $3 \le n_1 \le 5$ e *IO-5* y revelan la mejor conducta, con valores en un rango de 7.3% para Eil51 a 15.6% para KroD100. Nuevamente los peores valores son obtenidos bajo *IO-1* en el rango de 19.8% para Eil51 a 36.51% para Pr76.

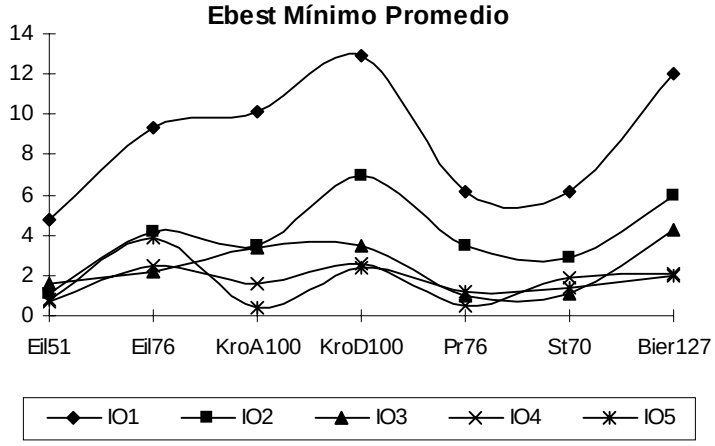

Figura 7.4. Promedio de valores de Ebest mínimo para todas las instancias bajo cada alternativa.

La figura 7.4, indica el promedio de los valores de Ebest mínimo para todas las instancias bajo cada alternativa. Los mejores valores se obtienen nuevamente con $3 \le n_1 \le 5$ en el rango de 0.41% (*IO-5* en KroA100) a 4.2% (*IO-3* en Bier127). *IO-5* brinda los mejores resultados en el rango de 0.41% para Kroa100 hasta 3.84% para Eil76. Mientras que *IO-1* obtiene los valores más desdeñables en un rango que va desde el 4.74% para eil51 a un 12.94% para KroD100.

La figura 7.5, indica el promedio general sobre todos los experimentos de la generación donde se encuentran los individuos mejor adaptados bajo cada alternativa. Como es de suponer, el mayor número de ciudades incrementa el número de generaciones necesarias para encontrar al mejor individuo en todos los algoritmos implementados. Nuevamente se puede detectar una mejor performance con $3 \le n_1 \le 5$. *IO-1*, es la opción que logra la performance más baja, con valores en el rango de las 532 generaciones en Eil51 a 1644 en Bier127 para encontrar la mejor solución. Mientras que todas las alternativas con

multirecombinación realizan una mejor tarea. *IO-4* e *IO-5* presentan los mejores desempeños; ellos necesitan desde 271 y 276 generaciones (en Eil51) hasta un total de 828 y 796 generaciones (en Bier127), respectivamente.

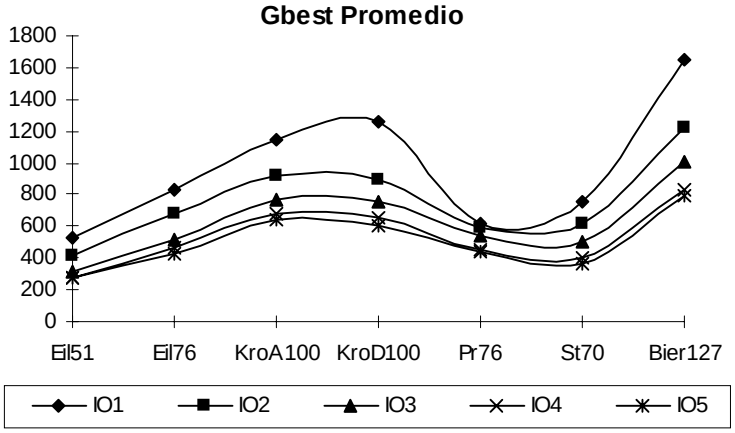

Figura 7.5. Promedio de los valores de Gbest para todas las instancias bajo cada alternativa.

7.5. CONCLUSIONES DE LOS EXPERIMENTOS

Las cuatro variantes con multirecombinación introducidas a la opción de inver-over, aplicadas al TSP: *IO-2 – IO-5*, permiten la múltiple aplicación del operador inver-over a cada individuo en la población. Lo cual se contrasta con el método original *IO-1*.

Todas los algoritmos evaluados siempre generan hijos factibles y se prueban para un conjunto de instancias seleccionadas del TSP.

A la luz de los resultados se puede concluir que:

- Considerando la calidad de los resultados (en los mejores individuos y en los individuos promedios) y la velocidad para encontrar soluciones muy cercanas a la óptima, todos los métodos que incluyen la multirecombinación superan al método original *IO-1*.
- Teniendo en cuenta todas las variables de performance Ebest, Epop, y Gbest antes analizadas, los valores son mejorados al incrementar el número n_1 de las operaciones de inver-over.

CONCLUSIONES GENERALES

Esta tesis presenta aplicaciones pertenecientes a las áreas ingenieriles, científicas e incluso administrativas (ej. redes, placas de circuitos impresos, planificación y programación de actividades en proyectos de desarrollo de sistemas), con características propias de los problemas de optimización combinatorial NP-Completos, especialmente relacionados con el problema del viajante de comercio (TSP). Como este problema es un miembro de la familia de los problemas NP-duros, los cuales no poseen un algoritmo que halle una solución en tiempo polinomial. Consecuentemente es posible no encontrar el óptimo global en algunas aplicaciones, o saber si la solución hallada es un óptimo global. En muchos casos puede ser preferible encontrar una muy buena solución en tiempo razonable, aunque esta no sea un óptimo global.

Los algoritmos tradicionales utilizados para solucionar los problemas de optimización combinatorial requieren un gran conocimiento previo de cómo evaluar el fitness (función de costo, superficie de respuesta, etc.). Este es el caso de la búsqueda basada en el gradiente que requiere una función de costo diferenciable. En cambio, los algoritmos evolutivos no necesitan conocer previamente el dominio del problema. Sólo necesita ordenar de acuerdo a un rango, a dos soluciones competidoras, esto es, determinar cuál solución es la más adecuada. Permitiendo de esta manera, que una amplia variedad de problemas de difícil tratamiento en el campo de la ingeniería puedan resolverse por medio de la opción evolutiva. En general, los algoritmos evolutivos son capaces de solucionar problemas que las técnicas numéricas no logran resolver.

Otra ventaja que ofrece la alternativa evolutiva es la capacidad para adaptarse a situaciones cambiantes. En los procedimientos de optimización tradicionales, el cálculo debe iniciarse desde cero si una de las variantes del problema cambia, lo que resulta desde el punto de vista computacional muy costoso. Con un algoritmo evolutivo la población final de soluciones actúa como una reserva de conocimiento que puede aplicarse a ambientes dinámicos. Sin necesidad de iniciar nuevamente el cálculo.

Básicamente los algoritmos evolutivos simulan la evolución de una población de soluciones para un problema dado. Dicho proceso consiste en: la evaluación de todos los individuos de la población, la selección de una nueva población intermedia, y el intercambio de su código genético. Estos algoritmos, son una clase de métodos de búsqueda de propósito general que combinan elementos de la búsqueda estocástica y dirigida; remarcando, así, el balance entre la exploración y la explotación del espacio de búsqueda. Cabe destacar que los algoritmos evolutivos pueden considerarse eficientes en comparación con otros métodos estocásticos de búsqueda ciega, pero en general: si se encuentra un método heurístico para resolver un problema específico este siempre lo hará mucho más eficientemente que un AE. En la práctica, es fundamental para el buen desempeño de un AE, controlar continuamente la diversidad de la población. No es conveniente poseer poca variedad de aptitudes ya que en tal caso todos los individuos tendrían más o menos las mismas posibilidades de sobrevivir, la selección reproduciría la situación anterior y todo el peso de la búsqueda recaería en los operadores genéticos, y acabaría siendo poco más que una búsqueda aleatoria. Además la población sería copada por superindividuos, que sólo son los más aptos en cierto momento, pero no las más aptos absolutos; provocando, en general, una *convergencia prematura* del AE, habitualmente hacia un subóptimo. Uno de los puntos clave en estos algoritmos es la codificación de la solución a un problema en un cromosoma.

Los AGs convencionales involucran la selección de dos individuos padres para crear a lo sumo dos cromosomas hijos. Pero la necesidad de mejorar la performance de los AGs en problemas de alta complejidad, ha llevado a Eiben, con su grupo, y a Esquivel, entre otros, reformular los mecanismos de selección y crossover. Tales cambios han dado origen a la multiplicidad de padres, de hijos y la combinación de ambas.

Muchos son los algoritmos, no evolutivos, para resolver el problema del viajante, algunos de estos proveen una solución óptima global para instancias no triviales de TSP con 7397 ciudades. Para lo cual necesitan 4 años de tiempo de CPU sobre una red de computadoras. En esta tesis se presenta como los algoritmos evolutivos, un método heurístico, es capaz de hallar un conjunto de soluciones promisorias velozmente. Los primeros avances para solucionar el TSP,

por medio de Algoritmos Evolutivos, necesitan desarrollar una apropiada representación para codificar un tour, operadores genéticos válidos para mantener los bloques evitando ilegalidad, y prevención de la convergencia prematura. El problema de los algoritmos evolutivos basados en operadores de crossover está en la necesidad de un mayor tiempo computacional, por lo que resultan muy caros. Mientras que los algoritmos basados en operadores de mutación no escapan de forma eficiente de los óptimos locales. Michalewicz propone un algoritmo evolutivo basado en el operador inver-over. El operador incorpora en una solución el conocimiento provisto por otros individuos, mediante la inversión y la recombinación. La inversión es aplicada a una parte de un único individuo, pero la selección del segmento a invertir depende de otros individuos pertenecientes a la población. Este algoritmo presenta las siguientes ventajas:

Es simple y fácil de implementar (menos de 100 líneas de código).

Los resultados experimentales indican que este operador obtiene mejores resultados que todos los otros operadores evolutivos propuestos para TSP (PMX, CX, OX, ER, EER).

Es bastante rápido y la calidad de sus resultados es muy alta.

Es posible hacer las siguientes observaciones:

Es probablemente el algoritmo evolutivo más rápido para el TSP. Todos los otros algoritmos basados en operadores de crossover proveen resultados mucho peores en un tiempo bastante más prolongado.

Tiene sólo 3 parámetros: el tamaño de la población, la probabilidad p de generar una inversión aleatoria, y el número de iteraciones en la condición de terminación; la mayoría de los otros sistemas evolutivos tienen muchos otros parámetros adicionales.

Introduce un nuevo e interesante operador, que combina características de inversión (o mutación) y crossover.

Con una nueva perspectiva a la naturaleza del operador inver-over, en esta tesis se introduce la característica de multiplicidad al mismo. Perimitiendo, de esta manera, la donación de información por parte de otros individuos de la población hasta que eventualmente un mejor vástago sea creado y reemplace a S_i en la población actual. Todas los algoritmos mediante los cuales se ha evaluado el TSP (desde IO-1 hasta IO-5), generan permanentemente hijos factibles. Además, al

evaluar la bondad de las soluciones y el escaso tiempo consumido en hallarlas muy cercanas al óptimo, se deduce que: las opciones que incluyen multirecombinación superan al método original de inver-over, y los valores son mejorados al incrementar el número n_1 de las operaciones de inver-over.

Resumiendo, en este trabajo se puede observar como la computación evolutiva, un método heurístico, brinda un grupo de diferentes soluciones a distintas instancias estáticas del problema del viajante de comercio, cuyo modelo responde al de aplicaciones en áreas tales como las redes de telecomunicaciones, la planificación de proyectos de desarrollo de cualquier índole, entre otras. No obstante, el hecho de contar con una batería de soluciones permite una rápida y adecuada adaptación de la aplicación ante cualquier modificación de las condiciones del problema. Por lo cual su utilidad puede extenderse fácil y económicamente a entornos dinámicos. Además, las técnicas de la computación evolutiva son accesibles y de fácil manejo para que el personal administrativo y técnico de una organización pueda resolver una amplia gama de situaciones problemáticas.

En función a la bondad de los resultados obtenidos, en trabajos futuros, se aspira a continuar investigando la aplicación del operador inver-over en un algoritmo evolutivo combinado con técnicas de búsqueda local; tales como tabu search, simulated annealing, entre otras. Ya sea tanto para distintas aplicaciones del problema del viajante de comercio como para las de problemas de scheduling por ejemplo, aquellos casos de scheduling con set-up dependiente de la secuencia, o problemas de flow shop.

REFERENCIAS

1. Aarts, E. H.L., *Simulated Annealing and Boltzman Machines*. Jhon Wiley, Chichester, UK, 1989.
2. Alfonso, H. Cesán P., Fernández, N., Minetti, G., Salto, C., Velzaco, L., Gallard, R., *Improving Evolutionary Algorithms Performance by Extending Incest Prevention*, en Proceedings del 4to Congreso Argentino de Ciencias de la Computación (CACiC'98), pag.323-334, Universidad Nacional del Comahue, Octubre 1998.
3. Agoston, E., Eiben, R., Michalewicz, Z., *Parameter Control in Evolutionary Algorithms*, Technical Report, UNC - Charlotte, USA, 1998.
4. Angeniol, B., Vaubois, G. D. L. C., Le Texier, J.-Y., *Self-organizing Feature Maps and the Travelling Salesman Problem*, Neural Networks 1, pag. 289-293, 1988.
5. Bäck T., Hoffmeister F., *Extended Selection Mechanisms in Genetic Algorithms*. Proceedings of the 4th Int. Conference on Genetic Algorithms, pag. 92-97. Belew, R., Booker, L., Editores, Morgan Kaufmann Publishers, 1991.
6. Bäck, T., *Selective Pressure in Evolutionary Algorithms, a Characterisation of Selection Mechanisms.* Proceeding of the First IEEE Conference on Evolutionary Computation, pag.57-62. IEEE Press, Fogel, D., Ed., 1994.
7. Bäck, T., *Evolutionary Algorithms in Theory and Practice*, Oxford University Press, New York, 1995.
8. Bäck, T., Schwefel, H., *Evolutionary Computation: an Overview*. Proceeding of the Third IEEE Conference on Evolutionary Computation, Fogel, D., Editor, IEEE Press, pag. 20-29, Nagoya, Japón, 1996
9. Bäck, T., *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*, Oxford University Press, New York, 1996.
10. Baker J. E., *Reducing Bias and Inefficiency in the Selection Algorithm*, Proceeding of the 2nd International Conference On Genetic Algorithms, pag. 14-21. Grefenstette, J.J., Ed., 1987.
11. Baker J. E., *Adaptive Selection Methods for Genetic Algorithms*, Proceeding of the 2nd International Conference On Genetic Algorithms, pag. 100-111. Grefenstette, J.J., Ed., 1987.
12. Bean J., *Genetic Algorithms and Random Keys for Sequencing and Optimisation*, ORSA Journal on Computing, vol. 6, no. 2, pag. 154-160, 1994.
13. Bennett, K., Ferris, M. C., Ioannidis, Y. E., *A Genetic for Database Query Optimization*, Proceeding of the 4th Int. Conference On Genetic Algorithms, pag. 400-407, Morgan Kaufmann, 1991.
14. Booker, L.B., *Intelligent Behaviour as an Adaptation to the Task Environment*, Doctoral Dissertion, Universidad de Michigan, 1982.
15. Booker, L.B., *Improving Search in Genetic Algorithms*, en Genetic Algorithms and Simulated Annealing, Davis, L. Editor. Morgan Kaufmann Publisers, Los Altos, CA, 1987.
16. Brindle, A., *Genetic Algorithms for Function Optimization*, Ph.D. thesis, Universidad de Alberta, Edmonton, 1981.
17. Burke, L. I., Damany, P., *The Guilty Net for The Traveling Salesman Problem*, Computers & Oper. Res 19, pag. 255-265, 1992.
18. Burkard, R., Deineko, V., Van Dal, R., Van der Veen, J., Woeginger, G., *Well-Solvable Special Cases of the TSP: A Survey*, Karl-Franzens-Universität Graz & Technische Universität Graz, Optimierung und Kontrolle, 1995.
19. Cerny, V., *A Thermodynamical Approach to the Travelling Salesman Problem: An Efficient Simulation Algorithm*, Journal of Optimization Theory and Application 45, pag. 41-51, 1985.
20. Cheng, R., Gen, M., *On Film-Copy Deliverer Problem*, en Zheng, W. editor, Proceedings of

the Second International Conference on Systems Science and Systems Engineering, pag. 542-547, Bejing, 1993.

21. Cheng, R., Gen, M., *Crossover on Intensive Search and Traveling Salesman Scheduling Problem*, en Fogel D., editor, Proceedings of the First IEEE Conference on Evolutionary Computation, IEEE Press, pag. 736-741, Orlando, FL, 1994.

22. Cheng, R., Gen, M., *Resource Constrained Project Scheduling Problem Using Genetic Algorithms*, International Journal of Intelligent Automation and soft Computation, 1996.

23. Croce, F., Roberto, T., Giuseppe, V., *A Genetic Algorithm for the Job Shop Problem*. Technical report, D.A.I. Politecnico di Torino. Computer and Operations Research, Italia, 1993.

24. CRPC, *Researchers Forge Optimal Path For Traveling Salesman Problem*, www.crpc.rice.edu, 1998.

25. Dasgupta, D., Michalewicz, Z., *Evolutionary Algorithms in Engineering Applications*. Springer, Springer-Verlag Berlin Heidelberg, Alemania, 1997.

26. Davis L., *Applying Adaptive Algorithms to Domains*, en proceedings of the International Joint Conference on Artificial Intelligence, pag. 162-164,1985.

27. Davis L., *Job Shop Scheduling with Genetic Algorithms*, Proceedings of the International Conference on Genetic Algorithms and their Applications, pag. 136-140, 1985.

28. Davis L., *Adapting Operator Probabilities in Genetic Algorithms*, Proceedings of the Third International Conference on Genetic Algorithms and their Applications, pag. 61-69, Schaffer, J., Editor, Morgan Kaufmann Publishers, San Mateo, CA, 1989.

29. Davis L., editor, *Handbook of Genetic algorithms*, Van Nostrand Reinhold, New York, 1991.

30. De Jong, K. A., *An Analysis of the Behaviour of a Class of Genetic Adaptive Systems*. Ph.D. Thesis, Universidad de Michigan, Ann Arbor, 1975.

31. De Jong, K. A., *Genetic Algorithms, A 10 Year PerspectiveK*, en Grefenstette J., editor, Proceedings of the First International Conference on Genetic Algorithms, Lawrence Erlbaum Associates, pag. 160-168, Hillsdales, NJ, 1985.

32. De Jong, K. A., Spears, W., *Using Genetic Algorithms to Solve NP-Complete Problems*. Proceedings of the Third International Conference on Genetic Algorithms and their Applications, pag. 133-139, Schaffer, J., Editor, Morgan Kaufmann Publishers, San Mateo, CA, 1989.

33. Durbin, R., Willshaw D., *An Analogue Approach to The Travelling Salesman Problem Using an Elastic Net Method*, Nature 326, pag. 689-691, 1987.

34. Eiben A.E., *A Method for Designing Decision Support Systems for Operational Planning*, PHD Thesis, Eindhoven University of Technology, 1991.

35. Eiben A.E., Raué P-E., Ruttkay Zs., *Genetic Algorithms with Multi-parent Recombination*, en Davidor, Schwefel H.-P., Männer R., editores, Proceedings of the 3rd Conference on Parallel Problem Solving from Nature, number 866 in LNCS, pag. 78-87. Springer-Verlag, 1994.

36. Eiben, A.E., van Kemenade, C.H.M, Kok, J.N., *Orgy in the Computer: Multi-parent Reproduction in Genetic Algorithms*, en Moran, F., Moreno, A., Merelo, J.J., Chacpn P., editores, Proceedings of the 3rd European Conference on Artificial Life, number 929 in LNAI, pag. 934-945, Springer-Verlag, 1995.

37. Eiben, A.E. y Bäck Th., *An Empirical Investigation of Multi-parent Recombination Operators in Evolution Strategies*, Evolutionary Computation. 5(3), pag 347-365, 1997.

38. Eiben, A.E, van Kemenade, C.H.M, *Diagonal crossover in Genetic Algorithms for Numerical Optimization*, en Journal of Control and Cybernetics, 26(3), pag. 447-465, 1997.

39. Eshelman, L.J., Caruna, A., Schaffer, J.D., *Biases in the Crossover Landscape*. Proceeding of the Third International Conference on Genetic Algorithms and their Applications, pag. 10-19,

Schaffer, J., Editor, Morgan Kaufmann Publishers, San Mateo, CA, 1989.

40. Eshelman, L.J., Schaffer, D., *Preventing Premature Convergence in Genetic Algorithms by Preventing Incest*, en Proceedings of the Fourth International Conference on Genetic Algorithms, pag. 115-122. Morgan Kauffman, San Mateo California, USA, 1991.

41. Eshelman, L.J., Schaffer, D., *Crossover´s Niche*, Proceedings of Fifth International Conference on Genetic Algorithms, Forrest, S. (Editor), pag. 9-14, Morgan Kaufmann, San Mateo, CA, 1993.

42. Eshelman, L.J., (Editor), *Proceedings of the Sixth International Conference on Genetic Algorithms*. Morgan Kaufmann, San Mateo, CA, 1995.

43. Esquivel, S., Gallard, R., Michalewicz, Z., *MCMP: Another Approach to Crossover in Genetic Algorithms*, Proceedings del Primer Congreso Argentino de Ciencias de la Computación, pag. 141-150, 1995.

44. Esquivel S., Leiva A., Gallard R., - *Multiple Crossover per Couple in Genetic Algorithms*, Proceedings of the Fourth IEEE Conference on Evolutionary Computation (ICEC'97), pag. 103-106, ISBN 0-7803-3949-5, Indianapolis, USA, Abril 1997.

45. Esquivel, S., Leiva, A. Gallard, R., *Couple Fitness Based Selection with Multiple Crossover per Couple in Genetic Algorithms*, Proceedings of the International Symposium on Engineering of Intelligent Systems (EIS'98), vol. 1, pag. 235-241, 1998.

46. Esquivel S., Leiva H., Gallard, R., *Self-Adaptation of Parameters for MCPC in Genetic Algorithms*, en Proceedings del 4to Congreso Argentino de Ciencias de la Computación (CACiC'98), pag. 419-425, Universidad Nacional del Comahue, Octubre 1998.

47. Esquivel S., Leiva H., Gallard R., *Multiple Crossovers Between Multiple Parents to Improve Search in Evolutionary Algorithms*, Proceedings of the 1999 Congress on Evolutionary Computation (IEEE), pag. 1589-1594, Washington DC, 1999.

48. Esquivel S., Leiva H., Gallard R., *Multiplicity in Genetics Algorithms to Face Multicriteria Optimization*, Proceedings of the 1999 Congress on Evolutionary Computation (IEEE), pag 85-90, Washington DC,1999.

49. EvoWeb, *Resources: Case Studies*, evonet.dcs.napier.ac.uk, 2000.

50. Fiechter, C.-N, *A Parallel Tabu Search Algorithm for Large Traveling Salesman Problems*, Disc. Applied Math 51, pag. 243-267, 1994.

51. Fogel, L. J., Owens, A. J., Walsh, M. J., *Artificial Intelligence through Simulated Evolution*, John Wiley, Chichester, UK, 1966.

52. Fogel, D.B., *Evolving Artificial Intelligence*. Ph.D. thesis, Universidad de California, San Diego, 1992.

53. Fogel, D.B., *An Introduction to Simulated Evolutionary Optimization*. IEEE transaction on Neural Networks, vol. 5, pag. 3-14, 1994.

54. Fogel, D.B., Stayton, L., *On the Effectiveness of Crossover in Simulated Evolutionary Optimization*, Biological Cybernetics, Vol. 32, pag. 171-182, 1994.

55. Fogel, D.B., *Evolutionary Computation, Toward a New Philosophy of Machine Intelligence*. IEEE Press. Piscataway, NJ, 1995.

56. Fogel, D, Ghozeil, A., *Using fitness distributions to Design more Efficient Evolutionary Computations*, Proceedings of the Third IEEE Conference on Evolutionary Computation, Fogel, D., Editor, IEEE Press, pag. 11-19, Nagoya, Japón, 1996

57. Fogel, L.J., Owens, A.J., Walsh, M. J., *Artificial Intelligence Thorough Simulated Evolution*. John Wiley, Chichester, UK, 1996.

58. Fogel, D., *What is Evolutionary Computation*, IEEE Spectrum Vol. 37 Nro. 2, pag. 26-32, 2000.

59. Forrest, S. (Editor), *Proceedings of Fifth International Conference on Genetic Algorithms*,

Morgan Kaufmann, San Mateo, CA, 1993.

60. Fox, B.R., McMahon, *Genetic Operators for Sequential Problems*, in the Foundations of Genetic Algorithms, Rawlins G., Editor, pag. 284 – 300, San Mateo, California, 1991.

61. Frantz, D.R., *Non Linearities in Genetic Adaptive Search*, Dissertation Abstracts International, 33(11), 5240B –5241B.

62. Gen M., Cheng R., *A survey of Penalty techniques in Genetic Algorithms*, en Proceeding of the Third IEEE Conference on Evolutionary Computation, Fogel, D., Editor, IEEE Press, pag. 804-809, Nagoya, Japón, 1996

63. Gen M., Cheng R., *Genetic Algorithms & Engineering Design*, Wiley Interscience, 1997.

64. Glover F., *Heuristics for Integer Programming Using Surrogate Constraints*, Decision Sciences, Vol. 8, Nro. 1, pag. 156-166, 1977.

65. Glover F., *Future Paths for Integer Programming and Links to Artificial Intelligence*, Computers & Ops. Res. 13, pag. 533-549, 1986.

66. Glover F., *Multilevel Tabu Search and Embedded Search Neighbourhoods for The Traveling Salesman Problem*, Manuscript, School of Business, University of Colorado, boulder, CO, 1991.

67. Glover F., *Ejection Chains, Reference Structures and Alternating Path Methods for The Traveling Salesman Problems*, Manuscript, School of Business, University of Colorado, boulder, CO, 192.

68. Goldberg, D., Ringle R., *Alleles, Loci and the Traveling Salesman Problem*, en Proceedings of the First Conference on Genetic Algorithms, Grefenstette J., editor, Lawrence Erlbaum Associates, pag. 154-159, Hillsdales, NJ, 1985

69. Goldberg, D., *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, Reading, MA, 1989.

70. Goldberg, D., Korb, B., Deb, K., *Messy Genetic Algorithms, motivation, analysis, and first results*, Complex Systems, vol. 3, pag. 493-530, 1989.

71. Goldberg D. E., Deb K., *A Comparison of Selection Schemes used in Genetic Algorithms*, Foundations of Genetic Algorithms I. Morgan Kaufmann, pag. 69-93.

72. Grefenstette J., Gopal R., Rosmaita B., Gucht D., *Genetic Algorithms for the Traveling Salesman Problem*, en Grefenstette J., editor, Proceedings of the First International Conference on Genetic Algorithms, Lawrence Erlbaum Associates, pag. 160-168, Hillsdale, NJ, 1985.

73. Grefenstette J., (Editor), *Proceedings of the First International Conference on Genetic Algorithms*, Lawrence Erlbaum Associates, Hillsdale, NJ, 1985

74. Grefenstette J., *Optimization of Control Parameters for Genetic Algorithms*, en IEEE transactions on Systems, Man and Cybernetics, SMC-16(1):122-128, 1986.

75. Grefenstette J., *Incorporating Problem Specific Knowledge into Genetic algorithms*, En Davis, L. (editor), Genetic Algorithms and Simulated Annealing, Morgan Kaufmann Publishers, San Mateo, CA, 1987.

76. Grefenstette J., (Editor), *Proceedings of the Second International Conference on Genetic Algorithms*, Lawrence Erlbaum Associates, Hillsdale, NJ, 1987

77. Grefenstette J., Baker, J., *How Genetic Algorithms work, a Critical Look at Implicit Parallelism*, en Proceeding of the Third International Conference on Genetic Algorithms, Schaffer, J., Editor, Morgan Kaufmann Publishers, San Mateo, CA, 1989.

78. Hamacher, H. W., *Combinatorial Optimization Problems Motivated by Robotic Assembly Problems*, en Akgul, M. Editor, Combinatorial Optimization, NATO ASI serie F82, pag. 187-198, 1992.

79. Heap, M., Kapur, R., Mourad, A., *A Fault Tolerant Implementation of The Traveling Salesman Problem*, Technical Report, Dept. of EECS, University of Texas, Austin, TX, 1989.

80. Hillier, F. S., Liberman, G.J., *Introduction to Operations Research*, Holden-Day, San Francisco, CA, 1967.

81. Hitomi, K., *Manufacturing Systems Engineering*, Second edition, Taylor & Francais, London, 1996.

82. Holland, J., *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, 1975.

83. Hopfield, J.J., Tank, D. W., *'Neural' Computation of Decisions in Optimization Problems*, Biol. Cybern 52, pag. 141-152, 1985.

84. Johnson, D.S., Aragon, C.R., McGeoch, L.A., Schevon C., *Optimization by Simulated Annealing: an Experimental Evaluation: Part I, Graph Partitioning*, Operations Research, 37(6), pag. 865-893, 1989.

85. Johnson, D.S., Aragon, C.R., McGeoch, L.A., Schevon C., *Optimization by Simulated Annealing: an Experimental Evaluation: Part II, Graph Colouring and Number Partitioning*, Operations Research, 39(3):378-406, 1991.

86. Johnson D., McGeoch L. *The Traveling Salesman Problem: A Case Study in Local Optimization*, Noviembre 20, 1995.

87. Jhonson, D.S., Aragon, C.R., McGeoch, L.A., y Schevon, C. *Optimization by Simulated Annealing: An Experimental Evaluation, part III*, Technical report, Bell Labs preprint.

88. Ilog, Inc., *Optimization Technology White Paper, A Comparative Study of Optimization Techniques*, www.ilog.com

89. Kirpatrick, S., Gelatt, C., Vecchi, M. *Optimization by Simulated Annealing*, Science, 1983.

90. Kirpatrick, S. *Optimization by simulated annealing: Quantitative Studies*, Journal Statistics Physical Science, 1984.

91. Knox, J., Glover F., *Comparative Testing of Traveling Salesman Heuristics Derived from Tabu Search, Genetic Algorithms and Simulated Annealing*, Technical Report, Center for Applied Artificial Intelligence, University of Colorado, 1989.

92. Knox, J., *Tabu Search Performance on the Symmetric Traveling Salesman Problem*, Computers & Ops. Res. 21, pag. 867-876, 1994.

93. Kohonen, T., *Self Organization and Associative Memory*, Springer-Verlag, Berlin, 1988

94. Koza, J. R., *Genetic Programming: A paradigm for Genetically Breeding Populations of Computer Programs to Solve Problems*, Reporte nro. STAN-CS-90-1314, Stanford University, 1990.

95. Koza, J. R., *Genetic Programming*, MIT Press, Cambridge, MA, 1992.

96. Koza, J. R., *Genetic Programming - 2*, MIT Press, Cambridge, MA, 1994.

97. In, L., Sikora, R., Shaw, M., *Joint Lot Sizing and Sequencing with Genetic Algorithms for Scheduling: Evolving the Chromosome Structure*, Beckman Institute for Advanced Science and Technology and Department of Business Administration University of Illinois at Urbana-Champaign, 1993.

98. Liepins, G., Hilliard, M., Pallmer, J., Morrow, M., *Greedy Genetics*, Proceedings of the Second International Conference on Genetic Algorithms, Lawrence Erlbaum Associates, pag. 90-99,Hillsdale, NJ, 1987.

99. Lin, S. *Computer solutions of The Traveling Salesman Problem*, Bell Syst. Tech. J., 1965.

100. Lin, S., Kernighan, B., *An Effective Heuristic Algorithm for the Traveling Salesman Problem*, Oper. Res., 1973.

101. Malek, M., Guruswamy, M., Pandya, M., *Serial and Parallel Simulated Annealing and Tabu Search algorithms for The Traveling Salesman Problem*, Ann. Operations Res. 21, pag. 59-84, 1989.

102. Marín, I., *Métodos de Exploración Dirigida*, Ediciones Macchi, Buenos Aires, Argentina, 1980.

103. Martin, C., Otto, S. *Combining Simulated Annealing with Local Search Heuristics, Metaheuristics in Combinatorial Optimization*, Editado por Laport G. y Osman I., Junio 14,1994.

104. Metropolis, N. A., Rosenbluth, M., Rosenbluth, A., Teller, A., Teller, E. *Equation of state calculations by fast computing machines*, Journal of Chemical Physics, 21:187--1092, 1953.

105. Michalewicz, Z., Krawczyk, j., Kazemi, M., Janikow, C., *Genetic Algorithms and Optimal Control Problems*, Proceeding of the 29th IEEE Conference on Decision and Control, Honolulu, pag.1664-1666, 1990.

106. Michalewicz, Z., *A Survey of Constraint Handling Techniques in Evolutionary Computation Methods*, Evolutionary Programming IV, McDonell, J., Reynolds, R., Fogel, D., Editores, MIT Press, Cambridge, MA, 1995.

107. Michalewicz, Z., *Genetic Algorithms + Data Structures = Evolution Programs*, Springer, third revised edition, 1996.

108. Michalewicz, Z., *Evolutionary Computation: Practical Issues*, Proceeding of the Third IEEE Conference on Evolutionary Computation, Fogel, D., Editor, IEEE Press, pag. 30-39, Nagoya, Japón, 1996

109. Michalewicz Z., Esquivel S., Gallard R., Michalewicz M., Tao G, Trojanowski, K., *The Spirit of Evolutionary Algorithms*, special issue on Evolutionary Computing of the Journal of Computing and Information Technology, University Computing Centre, Zagreb, Croatia, pag.1-18, 1999.

110. Minetti G., Gallard R., Alfonso H., *Inver-Over variants for de Euclidean Travelling Salesman Problem*, Proceedings of the Second International ICSC Symposium on Engineering Of Intelligent Systems, EIS'2000, Escocia, pag. 458-463, 2000.

111. Oliver I., Smith D., Holland J., *A study of Permutation Crossover Operators on the Traveling Salesman Problem*, in Grefenstette J. editor, Proceedings of the Second Conference on Genetic Algorithms, Lawrence Erlbaum Associates, pag. 224-230, Hillsdales, NJ, 1987.

112. Orvosh, D., Davis, L., *Using a Genetic Algorithm to Optimize Problems with Feasibility constraints*, Proceeding of the First IEEE Conference on Evolutionary Computation, Fogel, D., Ed., pag.548-552, IEEE Press. 1994.

113. Perez Serrada, A., *Una Introducción a la Computación Evolutiva*, 1996.

114. Rechenberg, I., *Evolutionsstrategie, Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*, Frommann-Holzboog Verlag, Stuttgart, 1973.

115. Reeves, C., *Diversity and Diversification in Algorithms: some Connections with Tabu Search*, Artificial Neural Nets and Genetic Algorithms, Albrecht, R., Reeves, C., Steele, N., Editores, Springer-Verlag, New York, 1993.

116. Reinelt, G., *TSPLIB 95*, Universität Heidelberg, Institut für Angewandte Mathematik, 1995.

117. Ronald, S., *Preventing Diversity Loss in a Routing Genetic Algorithm with Hash Tagging*, Complex Systems: Mechanism of Adaptation, Stonier, R., pag.133-140, Xing Huo Yu, editores, IOS Press, Amsterdam, 1994.

118. Ronald, S., *Genetic Algorithms and Scheduling Problems*, The Practical Handbook of Genetic Algorithms, vol. 1, pag. 397-430, L. Chambers editor, CRC Press, Boca Raton, Florida, 1995

119. Ronald, S., *Robust Encodings in Genetic Algorithms*, Evolutionary Algorithms in Engineering Application, pag. 29-44, Dasgupta D., Michalewicz, Z., editores, Springer, Springer-Verlag Berlin Heidelberg, Alemania, 1997.

120. Rossier, Y., Troyon, M., Liebling, T.M., *Probabilistic Exchange Algorithms and Euclidean Travelling Salesman Problems*, OR Spektrum 8, pag. 151-164, 1986.

121. Schaffer, J.D., Caruna, A., Eshelman, L.J., Das, R., *A Study of Control Parameters Affecting Online Performance of Genetic Algorithms for Function Optimization*, Proceedings of the Third International Conference on Genetic Algorithms, pag. 51-60, Scahffer, J. Editor, Morgan Kaufmann Publishers, San Mateo, CA, 1989.

122. Scahffer, J. (Editor), *Proceedings of the Third International Conference on Genetic Algorithms*, Morgan Kaufmann Publishers, San Mateo, CA, 1989.

123. Scahffer, J., Eshelman L., *On Crossover as an Evolutionary Viable Strategy*, Proceedings of the Fourth International Conference on Genetic Algorithms, pag. 61-68, 1991.

124. Schwefel, H. –P., *Evolutionsstrategie und numerische Optimierung*, Dissertation, Universidad Técnica de Berlin, 1975.

125. Schwefel, H. –P., *Numerische Optimierung von Compute-Modellen mitels der Evolutionsstrategie*, Vol. 26 de Interdisciplinary Systems Research, Birkhäuser, Basel, 1977.

126. Schwefel, H. –P., *Numerical Optimization of Computer Models*, John Wiley, Chichester, UK, 1981.

127. Schwefel, H., *Collective Phenomena in evolutionary Systems*, en Preprints of the 31st Annual Meeting of the International Society for General System Research, Budapest, Vol. 2, pag. 1025-1033, 1987.

128. Schwefel, H., *Evolution and Optimum Seeking*, John Wiley & Sons, New York, 1994.

129. Secretaría de Desarrollo Urbano y Ecología- Subsecretaría de Ecología, Dirección General de Prevención y Control Ambiental, *Especificaciones Técnicas para la Elaboración de Proyectos Ejecutivos de Manejo y Disposición Final de Residuos Sólidos Municipales*, México,1985.

130. Singh, N., *Systems Approach to Computer-Integrated Design and Manufacturing*, John Wiley & Sons, New York, 1996.

131. Smith, A., Tate, D., *Genetic Optimization using a Penalty Function*, Proceedings of Fifth International Conference on Genetic Algorithms, Forrest, S. (Editor), pag. 499-505, Morgan Kaufmann, San Mateo, CA, 1993.

132. Sandia National Laboratories, *Branch and Bound*, en Global Optimization Survey - Main Page, www.sandia.gov, 1997.

133. Snowdon, G., *Genetic Algorithms*, Aimaze,

www.personal.usyd.edu.au/~desm/afc-ga.html, 1996.

134. Spears W. M., *A Study of Crossover Operators in Genetic Programming*, en International Symposium on Methodologies for Intelligent Systems, pag. 409-418, 1991.

135. Spears W. M., *Adapting Crossover in Evolutionary Algorithms*, Proceedings of the Evolutionary Programming Conference, 1995.

136. Stansfield, W., *Theory and Problems of Genetics*, McGrawHill, 1991.

137. Syswerda, G., *Uniform Crossover in Genetic Algorithms*, Proceeding of the Third International Conference on Genetic Algorithms, Schaffer, J., Editor, pag. 2-9. Morgan Kaufmann Publishers, San Mateo, CA, 1989.

138. Tanenbaum, A. S., *Redes de Computadoras*, tercera edición, editorial Prentice-Hall Hispanoamericana, S.A., 1997.

139. Tao G., Michalewicz Z., *Inver-over Operator for the ETSP*, Proceedings of the 5th Parallel Problem Solving from Nature, T. Bäck, A. Eiben, M. Schoenauer, H-P. Schwefel (Editores), Amsterdam, September 1998, pag. 803-812.

140. Thierens, D., Goldberg, D., *Convergence Models of Genetic Algorithms Selection Schemes*, Parallel Problem Solving from Nature: PAG.SN III, Davidor, Y., Schwefel, H., Männer, R., editores, pag. 119-129, Springer-Verlag, Berlín, 1994.

141. Troyon, M., *Quelques Heuristics et Résultats Asymptotiques Pour Trois Problems*

d´Optimiastion Combinatoire, These Nro. 754, Ecole Polytechnique Federale de Lausanne, Lausane, Suiza, 1988.

142. van Laarhoven P., J. M., Aarts, E. H. L. *Simulated Annealing: Theory and Applications*, Mathematics and its applications, D. Reidel Publishing Company, Dordrecht, Holland, 1987.

143. Vignaux, G. A., Michalewicz Z., *A Genetic Algorithm for the Transportation Problem*, Proceedings of the 4th International Symposium on Methodologies for Intelligent Systems, North-Holland, Amsterdam, pag 252-259, 1989.

144. Vignaux, G. A., Michalewicz Z., *A Genetic Algorithm for the Linear Transportation Problem*, IEEE Transactions on Systems, Man and Cybernetics, Vol. 21, Nro. 2, pag. 445-452, 1991.

145. Whetzel, A., *Evaluation of the Effectiveness of Genetic Algorithms on Combinatorial Optimization*, Technical Report, Universidad de Pittsburgh, 1983.

146. Whitley, D., Starkweather, T., Fuquay, D'A., *Scheduling Problems and Traveling Salesman, The Genetic Edge Recombination Operator*, in the Proceedings of the Third International Conference Genetic Algorithms, Morgan Kaufmann Publishers, pag. 133–140, San Mateo, CA, 1987.

147. Whitley, D., *GENITOR: A Different Genetic Algorithm*, en Proceedings of the Rocky Mountain Conference on Artificial Intelligence, Denver, 1989.

148. Whitley, D., Starkweather, T., Shaner, D., *The Traveling Salesman and Sequence Scheduling: Quality Solutions Using Genetic Edge Recombination*, en Davis, L. Handbook of Genetic Algorithms, Van Nostrand Reinhold, New York, 1991.

149. Yamamura M., Ono T., Kobashi S., *Character-preserving Genetic Algorithms for Travelling Salesman Problem*, Journal of Japan Society for Artificial Intelligence, vol. 6, pag. 1049-1059, 1992.

150. Yamamura M., Ono T., Kobashi S., *Emergent search on double circle TSP using Subtour Exchange Crossover*, Proceedings of the Third IEEE Conference on Evolutionary Computation, Fogel D., editor, IEEE press, pag. 535-540, Nagoya, Japan, 1996.

151. Zhang, L., Zheng, W., *Remodeling the Film-Copy Deliverer Problem*, en Proceedings of IEEE International Conference on Systems, Man, and Cybernetics, pag. 543-547, San Antonio, 1994.