ÍNDICE

1. INTRODUCCIÓN

Desde que los ordenadores fueron inventados, nos hemos preguntado qué son capaces de aprender. Si pudiéramos entender cómo podemos programarlos para aprender (a que mejoren automáticamente a través de la experiencia) el impacto sería muy grande.

Imaginemos que existen ordenadores capaces de descubrir por medio del aprendizaje tratamientos médicos más efectivos contra nuevas enfermedades, casas inteligentes que aprenden a través de la experiencia de optimizar costes energéticos del uso particular de sus ocupantes, o ayudantes personales de software que aprenden de los intereses de sus usuarios para ser capaces de subrayar los sucesos especialmente relevantes del periódico diario en internet.

El hecho de descubrir cómo hacer que los ordenadores 'entiendan' haría que aparecieran nuevos usos de los ordenadores y nuevos niveles de competencia y un entendimiento detallado por parte de éstos de la información y el procesamiento de algoritmos mediante el aprendizaje de máquina nos conduciría a un mejor entendimiento de las habilidades e incapacidades del aprendizaje humano.

Todavía no sabemos cómo lograr que los ordenadores aprendan al mismo nivel que lo hacen los humanos. Sin embargo, se han inventado algunos algoritmos que son efectivos con cierto tipo de tareas de aprendizaje y un entendimiento teórico de dicho aprendizaje está comenzando a emerger. Se han desarrollado algunos programas prácticos de ordenador para exponer tipos útiles de aprendizaje y han empezado a aparecer aplicaciones comerciales significativas. Todos los acercamientos intentados hasta la fecha han sido para resolver problemas tales como el reconocimiento de voz y la búsqueda de algoritmos basados en el aprendizaje de máquina.

En el campo conocido como data mining (exploración de datos) se han comenzado a usar rutinariamente algoritmos de aprendizaje de máquina para descubrir valiosos conocimientos sobre grandes bases de datos comerciales en temas como aplicaciones en préstamos, transacciones financieras, avances médicos, etc. Mientras nuestro propio entendimiento de los ordenadores continúa madurando, parece inevitable que el aprendizaje de máquina juegue un papel central en tecnología y ciencia computacional.

Como ejemplo, se han desarrollado exitosamente programas que aprenden a reconocer palabras habladas (Waibel, 1989; Lee, 1989), a predecir el porcentaje de recuperación de pacientes que padecen neumonía (Cooper, 1997), a detectar el uso fraudulento de tarjetas de crédito, a conducir vehículos automáticamente en carreteras públicas (Pomerleau, 1989), y a ensayar juegos tales como el backgammon (juego de mesa para dos jugadores que en tiempos

del rey Alfonso X de Castilla era conocido con el nombre de juego de tablas) en niveles que se acercan a la altura de los campeones del mundo (Tesauro, 1992, 1995).

Se han desarrollado resultados teóricos para caracterizar la relación fundamental entre el número de ejemplos entrenados que se han observado, el número de hipótesis bajo consideración y el error esperado en las hipótesis aprendidas.
Hoy en día estamos comenzando a obtener modelos iniciales de aprendizaje humano y animal y a entender su relación con los algoritmos de aprendizaje realizados por las computadoras.

ALGUNAS APLICACIONES RECIENTES DEL APRENDIZAJE DE MÁQUINA:

Como principales aplicaciones del aprendizaje de máquina que se han desarrollado recientemente cabe destacar las siguientes:

- *Reconocimiento de palabras habladas*

  Los sistemas de reconocimiento de voz más exitosos emplean la misma base del aprendizaje de máquina. Por ejemplo, el sistema SPHINX (Lee, 1989) es capaz de aprender estrategias de habla específicas para reconocer sonidos primitivos (fonemas) y también palabras de la señal de voz escuchada. Se usan métodos de aprendizaje con redes neuronales y con modelos de Markov para acostumbrar a la máquina a la voz humana, vocabularios, características de micrófono, ruido de fondo, etc. Técnicas similares tienen potentes aplicaciones en problemas de interpretación de datos.

- *Conducción automática de un vehículo*

  Se han utilizado métodos de aprendizaje de máquina para enseñar a vehículos controlados por ordenador a conducir correctamente a través de una gran variedad de tipos de vías. Por ejemplo, el sistema ALVININ (Pomerleau, 1989) ha empleado estrategias aprendidas para conducir un vehículo sin asistencia a 70 millas/hora a lo largo de 90 millas en carreteras públicas y entre otros coches. Técnicas similares tienen posibles aplicaciones en problemas de control mediante el uso de sensores.

- *Clasificación de nuevas estructuras astronómicas*

  Se han aplicado métodos de aprendizaje de máquina a gran variedad de grandes bases de datos para poder analizar regularidades generales implícitas en los propios datos. Por ejemplo, la NASA ha usado algoritmos de aprendizaje mediante árboles de decisión para aprender a clasificar objetos celestiales desde el observatorio Palomar (Fayyad, 1995). Este sistema se está utilizando actualmente para clasificar automáticamente todos los objetos

del cielo, el cual es posible representar utilizando 3 terrabytes de datos de imagen.

❖ *Aprender a jugar al clásico mundial Backgammon*

Los programas de ordenador más exitosos para jugar a juegos similares al backgammon están basados en algoritmos de aprendizaje de máquina. Por ejemplo, el principal programa de ordenador para jugar a este juego llamado TD-GAMMON (Tesauro, 1992, 1995), aprende sus propias estrategias jugando sobre un millón de juegos contra sí mismo. Hoy en día dicho programa compite contra el campeón mundial. Similares técnicas tienen aplicaciones en algunos problemas prácticos en los que hay grandes espacios de búsqueda que deben ser examinados eficientemente.

❖ *Otras aplicaciones*

- Inteligencia artificial (problemas de búsqueda, resolución de problemas, …).
- Métodos bayesianos (el teorema de Bayes como base para el cálculo de probabilidades).
- Teoría de la complejidad computacional.
- Teoría de control (optimización de objetivos predefinidos y aprendizaje de la predicción del siguiente estado del proceso que se está controlando).
- Teoría de la información (medidas de la entropía e información contenida).
- Filosofía (teoría basada en que la hipótesis más simple siempre es la mejor).
- Psicología (ley del poder de la práctica) y neurobiología (redes neuronales).
- Estadística (errores en la estimación, intervalos de confianza, contrastes de hipótesis…).

LOS ALGORITMOS GENÉTICOS COMO SISTEMA DE APRENDIZAJE:

Los algoritmos genéticos proporcionan un acercamiento al aprendizaje basado aproximadamente en la evolución simulada. Los individuos son descritos la mayoría de las veces mediante cadenas de bits cuya interpretación depende de la aplicación, aunque dichos individuos también pueden ser descritos por expresiones simbólicas e incluso por programas de ordenador.

La búsqueda de un individuo apropiado comienza con una población (o colección) de individuos iniciales. Los miembros de una población actual dada, alcanzan la siguiente generación mediante operaciones tales como la mutación y el cruzamiento aleatorios (que están emparentados con los procesos de la evolución biológica).

En cada paso, los individuos de la población actual son evaluados en relación a una medida de adaptación dada (función fitness o función de aptitud), seleccionando probabilísticamente los individuos más adecuados para que pasen a formar parte de la siguiente generación.

Los algoritmos genéticos han sido aplicados exitosamente en gran variedad de tareas de aprendizaje y en problemas de optimización. Por ejemplo, han sido utilizados en el aprendizaje de reglas para el control robótico y para optimizar la topología y el aprendizaje de parámetros de redes neuronales artificiales.

1.1 HISTORIA DE LOS ALGORITMOS GENÉTICOS

Los primeros ejemplos de lo que hoy podríamos llamar algoritmos genéticos aparecieron a finales de los 50 y principios de los 60, programados en computadoras por biólogos evolutivos que buscaban explícitamente realizar modelos de aspectos de la evolución natural. A ninguno de ellos se le ocurrió que esta estrategia podría aplicarse de manera más general a los problemas artificiales, pero ese reconocimiento no tardaría en llegar: ``La computación evolutiva estaba definitivamente en el aire en los días formativos de la computadora electrónica" (Mitchell 1996).

En 1962, investigadores como G.E.P. Box, G.J. Friedman, W.W. Bledsoe y H.J. Bremermann habían desarrollado independientemente algoritmos inspirados en la evolución para optimización de funciones y aprendizaje automático, pero sus trabajos generaron poca reacción.

En 1965 surgió un desarrollo más exitoso, cuando Ingo Rechenberg, entonces de la Universidad Técnica de Berlín, introdujo una técnica que llamó estrategia evolutiva, aunque se parecía más a los trepacolinas que a los algoritmos genéticos. En esta técnica no había población ni cruzamiento; un padre mutaba para producir un descendiente, y se conservaba el mejor de los dos, convirtiéndose en el padre de la siguiente ronda de mutación. Versiones posteriores introdujeron la idea de población. Las estrategias evolutivas todavía se emplean hoy en día por ingenieros y científicos, sobre todo en Alemania.

El siguiente desarrollo importante en el campo vino en 1966, cuando L.J. Fogel, A.J. Owens y M.J. Walsh introdujeron en América una técnica que llamaron programación evolutiva. En este método, las soluciones candidatas para los problemas se representaban como máquinas de estado finito sencillas; al igual que en la estrategia evolutiva de Rechenberg, su algoritmo funcionaba mutando aleatoriamente una de estas máquinas simuladas y conservando la mejor de las dos (Mitchell 1996; Goldberg 1989). También al igual que las estrategias evolutivas, hoy en día existe una formulación más amplia de la técnica de programación evolutiva que todavía es un área de investigación en curso.

Fue John Holland en los años 60 quien en su estudio sobre sistemas adaptativos estableció las bases para desarrollos posteriores. Holland desde pequeño se preguntaba cómo logra la naturaleza crear seres cada vez más perfectos. Lo curioso era que todo se lleva a cabo a base de interacciones locales entre individuos, y entre éstos y lo que les rodea. No sabía la respuesta, pero tenía una cierta idea de cómo hallarla: tratando de hacer pequeños modelos de la naturaleza, que tuvieran alguna de sus características, y ver cómo funcionaban, para luego extrapolar sus conclusiones a la totalidad. De hecho, ya de pequeño hacía simulaciones de batallas célebres con todos sus elementos: copiaba mapas y los cubría luego de pequeños ejércitos que se enfrentaban entre sí.

En los años 50 entró en contacto con los primeros ordenadores, donde pudo llevar a cabo algunas de sus ideas, aunque no se encontró en un ambiente intelectual fértil para propagarlas. Fue en 1962, en la Universidad de Michigan en Ann Arbor, donde, dentro del grupo *Logic of computers*, sus ideas comenzaron a desarrollarse y a dar frutos. Y fue, además, leyendo un libro escrito por un biólogo evolucionista, R.A. Fisher, titulado *La teoría genética de la selección natural,* como comenzó a descubrir los medios de llevar a cabo sus propósitos de comprensión de la naturaleza. De este libro aprendió que la evolución era una forma de adaptación más potente que el simple aprendizaje, y tomó la decisión de aplicar estas ideas para desarrollar programas bien adaptados para un fín determinado.

En esta universidad, Holland impartía un curso titulado *Teoría de los sistemas adaptativos*. Dentro de este curso, y con una participación activa por parte de sus estudiantes, fue donde se crearon las ideas que más tarde se convertirían en los algoritmos genéticos.

Por tanto, cuando Holland se enfrentó a los algoritmos genéticos, los objetivos de su investigación fueron dos:

- Imitar los procesos adaptativos de los sistemas naturales, y
- Diseñar sistemas artificiales (normalmente programas) que retengan los mecanismos importantes de los sistemas naturales.

Unos 15 años más adelante, David Goldberg, actual delfín de los algoritmos genéticos, conoció a Holland, y se convirtió en su estudiante. Goldberg era un ingeniero industrial trabajando en diseño de *pipelines*, y fue uno de los primeros que trató de aplicar los algoritmos genéticos a problemas industriales. Aunque Holland trató de disuadirle porque pensaba que el problema era excesivamente complicado como para aplicarle algoritmos genéticos, Goldberg consiguió lo que quería, escribiendo un algoritmo genético en un ordenador personal Apple II. Estas y otras aplicaciones creadas por estudiantes de Holland convirtieron a los algoritmos genéticos en un campo con base suficiente aceptado para celebrar la primera conferencia en 1985 ICGA'85. Tal conferencia se sigue celebrando bianualmente.

John Henry Holland

David Goldberg

1.2. MECANISMOS BIOLÓGICOS UTILIZADOS PARA EL DESARROLLO DE LOS ALGORITMOS GENÉTICOS

En esta sección hablaremos de los mecanismos de la herencia con el objetivo de evidenciar la analogía entre los algoritmos genéticos y aquello que pretenden simular.

Cada individuo de cada una de las especies del planeta posee ciertas características que lo identifican. Dichas características, que suelen ser externas (como el color de ojos) aunque también pueden ser internas (como el grupo sanguíneo), forman el fenotipo de un individuo que es el resultado de la interacción del medio ambiente en que se desarrolla y la herencia que recibe de sus ancestros. Dicho fenotipo está determinado por las proteínas que produce y está definido en la información genética de cada una de las células del individuo. A su vez, la información acerca de las proteínas que se producirán está contenida en los cromosomas. En cada célula existen dos juegos de cromosomas que definen las mismas características. Un cromosoma es una larga molécula de ADN formada por proteínas que definen los genes. El conjunto de todos los cromosomas, es decir, de toda la información genética de un individuo se llama genoma y el conjunto de genes contenidos en el genoma se denomina genotipo que es que determina en buena medida el fenotipo del individuo.

Existen unas células especiales, llamadas gametos, que intervienen en la reproducción dividiéndose mediante un proceso denominado meiosis del siguiente modo:

1) Se duplica el número de cromosomas en la célula, esto es, se hace una copia de cada cromosoma. Al final quedan dos juegos correspondientes al padre y dos a la madre.

2) Se cruzan un juego de cromosomas del padre con uno de la madre, formándose dos juegos de cromosomas híbridos. El resultado es un

juego de cromosomas puros del padre, un juego puro de la madre y dos juegos de cromosomas híbridos.

3) Se divide la célula dos veces y al final del proceso quedan cuatro células haploides (con un solo juego de cromosomas): una con cromosomas puros del padre, una con cromosomas puros de la madre y dos con cromosomas híbridos.

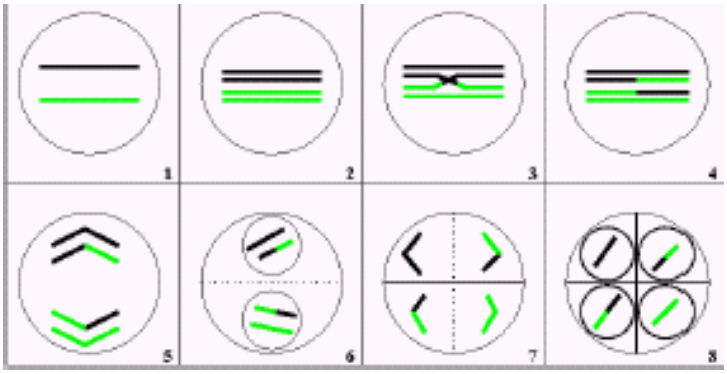

Ilustración 1: proceso de meiosis

Para el cruzamiento de dos cromosomas se forman entre ellos puntos de ruptura y unión de las cadenas de ADN que se denominan quiasmas y cortan el cromosoma en segmentos llamados cromáticas y unen cromáticas complementarias de dos cromosomas distintos.

Para la réplica un cromosoma ya existente interviene una enzima denominada ADN-polimerasa. La molécula de ADN tiene forma de doble hélice y dicha enzima se encarga de abrir por la mitad la molécula y replicar el cromosoma. Ocasionalmente la ADN-polimerasa comete un error que puede ser causado por radiaciones energéticas externas o sustancias extrañas. La alteración de dicha molécula de ADN constituye una mutación que puede manifestarse en el fenotipo y hacer al individuo diferente del resto de sus congéneres. Por lo general las mutaciones son desfavorables e incluso letales para el organismo mutante pero a veces pueden no serlo y conferir a dicho organismo alguna ventaja que le permita sobrevivir más fácilmente en su medio. Dicha característica será transmitida a sus descendientes y se habrá producido un pequeño paso evolutivo.

2. DEFINICIÓN Y PROCESO DE CONSTRUCCIÓN. ALGORITMO GENÉTICO SIMPLE Y ALGORITMO GENÉTICO PARALELO

Los algoritmos genéticos (AGs) son métodos adaptativos que pueden usarse para resolver problemas de búsqueda y optimización. Están basados en el proceso genético de los organismos vivos. A lo largo de las generaciones, las poblaciones evolucionan en la naturaleza de acuerdo con los principios de la selección natural y la supervivencia de los más fuertes, postulados por Darwin. Por imitación de este proceso, los algoritmos Genéticos son capaces de ir creando soluciones para problemas del mundo real. La evolución de dichas soluciones hacia valores óptimos del problema depende en buena medida de una adecuada codificación de las mismas. No sería conveniente la utilización de algoritmos genéticos en las siguientes situaciones:

- Si se requiere llegar forzosamente al máximo global en el caso de que estemos maximizando una función.
- Si conocemos la función de optimización.
- Si el problema está muy delimitado y se presta a un tratamiento analítico (funciones de una variable).
- Si la función es suave y convexa.
- Si el espacio es limitado, entonces es mejor enumerar todas las soluciones posibles.

2.1. DEFINICIÓN DE ALGORITMO GENÉTICO

Un **algoritmo genético** consiste en una función matemática o una rutina de software que toma como entradas a los ejemplares y retorna como salidas cuales de ellos deben generar descendencia para la nueva generación.

Algunas versiones más complejas de algoritmos genéticos generan un ciclo iterativo que directamente toma a la especie (el total de los ejemplares) y crea una nueva generación que reemplaza a la antigua una cantidad de veces determinada por su propio diseño. Una de sus características principales es la de ir perfeccionando su propia heurística en el proceso de ejecución, por lo que no requiere largos períodos de entrenamiento especializado por parte del ser humano, principal defecto de otros métodos para solucionar problemas, como los Sistemas Expertos.

La aplicación más común de los algoritmos genéticos ha sido la solución de problemas de optimización, en donde han mostrado ser muy eficientes y

confiables. Sin embargo, no todos los problemas pudieran ser apropiados para la técnica, y se recomienda en general tomar en cuenta las siguientes características del mismo antes de intentar usarla:

- Su espacio de búsqueda (i.e., sus posibles soluciones) debe estar delimitado dentro de un cierto rango.
- Debe poderse definir una función de aptitud (función fitness) que nos indique cómo es de buena o mala es una cierta respuesta.
- Las soluciones deben codificarse de una forma que resulte relativamente fácil de implementar en la computadora.

El primer punto es muy importante, y lo más recomendable es intentar resolver problemas que tengan espacios de búsqueda discretos aunque éstos sean muy grandes. Sin embargo, también podrá intentarse usar la técnica con espacios de búsqueda continuos, pero preferentemente cuando exista un rango de soluciones relativamente pequeño.

La **función de aptitud** no es más que la función objetivo de nuestro problema de optimización. El algoritmo genético únicamente maximiza, pero la minimización puede realizarse fácilmente utilizando el recíproco de la función maximizante (debe cuidarse, por supuesto, que el recíproco de la función no genere una división por cero). Una característica que debe tener esta función es que tiene ser capaz de "castigar" a las malas soluciones, y de "premiar" a las buenas, de forma que sean estas últimas las que se propaguen con mayor rapidez.

La **codificación** más común de las soluciones es a través de cadenas binarias, aunque se han utilizado también números reales y letras. El primero de estos esquemas ha gozado de mucha popularidad debido a que es el que propuso originalmente Holland, y además porque resulta muy sencillo de implementar.

2.2. PROCESO DE CONSTRUCCIÓN DE UN ALGORITMO GENÉTICO

El algoritmo genético simula el proceso de evolución natural. En esta sección intentaremos aclarar la manera como se lleva a cabo esta simulación.

❖ *Codificación del dominio*

En la naturaleza las características de los seres vivos, incluso aquéllas que los hacen óptimos para habitar en su medio, están determinadas por las proteínas que producen, las cuales se codifican en el material genético contenido en cada una de las células del individuo. Así pues, la naturaleza ha codificado el dominio del problema (todos los posibles individuos) aplicándolo al conjunto de todas las

posibles secuencias de nucleótidos (que son los elementos que forman las proteínas).

Así, para un algoritmo genético lo primero que se requiere es determinar en qué espacio se encuentran las posibles soluciones al problema que se pretende resolver. En caso de tener un problema de optimización de una función cuyo dominio es un subconjunto de números reales, entonces este subconjunto es al que nos referimos. Es necesario codificar de alguna manera el dominio del problema para obtener estructuras manejables que puedan ser manipuladas por el algoritmos genético. Cada una de estas estructuras constituye el equivalente al genotipo de un individuo en términos biológicos. El elemento del dominio del problema al que se mapea este genotipo es el análogo al fenotipo. Es frecuente que el código de los elementos del dominio del problema utilice un alfabeto binario (0's y 1's).

Una vez que se ha definido la manera de codificar los elementos del dominio del problema y se conoce la forma de pasar de un elemento a su código y viceversa, es necesario fijar un punto de partida. Los algoritmos genéticos manipulan conjuntos de códigos (poblaciones de códigos) en generaciones sucesivas. El algoritmo se encargará de favorecer la aparición en la población de códigos que correspondan a elementos del dominio que estén próximos a resolver el problema. Es decir, dicho algoritmo recibirá como entrada una población de códigos y a partir de ésta generará nuevas poblaciones, donde algunos códigos desaparecerán mientras que otros, que se mapean en mejores soluciones posibles, aparecen con más frecuencia hasta que se encuentra una satisfactoria o hasta que se cumpla alguna otra condición de terminación. Los elementos de la población serán llamados individuos y a los códigos se les denominará indistintamente cromosomas, genotipo, genoma o código genético.

❖ *Evaluación de la población*

En la naturaleza hay individuos más hábiles que otros para sobrevivir y al igual que sucede en la naturaleza, en los algoritmos genéticos es necesario establecer algún criterio que permita decidir cuáles de las soluciones propuestas en una población son mejores respecto del resto de las propuestas y cuáles no lo son.

Para determinar cuáles de estos individuos corresponden a buenas propuestas de solución y cuáles no, es necesario calificarlos de alguna manera. Cada individuo de cada generación de un algoritmo genético recibe una calificación o una medida de su grado de adaptación (fitness). Éste es un número real no negativo que será más grande cuanto mejor sea la solución propuesta por dicho individuo. Por ejemplo, si el problema a resolver consiste en maximizar una función, entonces la calificación asignada a un individuo determinado debe indicar qué tan alto es el valor de la función en el elemento de su dominio codificado por el individuo. Si, en cambio, el problema es determinar la ruta más

corta entre dos puntos, la calificación deberá ser tanto más alta cuanto más corto sea el camino codificado en el individuo que esté siendo calificado.

Al hablar de que a cada individuo de la población se le asigna una y sólo una calificación, se está hablando de una función que se denomina función de adaptación, cuya evaluación puede no ser sencilla y es, de hecho, lo que en la mayoría de los casos consume más tiempo en la ejecución de un algoritmo genético. Hay que tener en cuenta que se evalúa una vez en cada individuo en cada generación. Si un AG es ejecutado con una población de tamaño 100 durante 100 generaciones, la función es evaluada 10000 veces. Además, puede darse el caso de que la función de evaluación no tenga una regla de correspondencia explícita, esto es, una expresión algebraica, y puede ocurrir incluso que la función cambie de generación en generación.

❖ *Selección*

Una vez calificados todos los individuos de una generación, el algoritmo debe seleccionar a los individuos más calificados para que tengan mayor oportunidad de reproducción. De esta forma se incrementa la probabilidad de tener individuos 'buenos' en el futuro. Si de una determinada generación se seleccionaran sólo aquellos con una calificación mayor o igual que cierto número *c* para pasarlos a la siguiente generación, es claro que en ésta la calificación superará *c* y por tanto al promedio de la generación anterior. La selección explota el conocimiento que se ha obtenido hasta el momento, procurando elegir lo mejor que se haya encontrado, elevando así el nivel de adaptación de toda la población.

En principio podría parecer que es conveniente tener una estrategia de selección estricta para que se mejore rápidamente la población y converja el algoritmo, es decir, que la población se acumule alrededor de un genotipo óptimo. Esto puede no ser cierto, ya que podría ocurrir que la población se acumulara rápidamente alrededor de algún individuo que sea bueno, comparativamente con el resto de los individuos considerados a lo largo de ejecución del algoritmo, pero este individuo puede no ser el mejor posible. A esto se le suele llamar convergencia prematura y no se puede asegurar pero sí procurar que lo anterior no ocurra. Además de la explotación es necesario que exista exploración. El AG debe, no sólo seleccionar de entre lo mejor que ha encontrado, sino procurar encontrar mejores individuos. A esto se dedican los operadores que se describirán a continuación (cruzamiento y mutación).

En la estrategia de selección normalmente se incluye un elemento extra que sirve de 'ancla'. Si sólo se hace selección forzando que sea más probable elegir al mejor individuo de la población pero sin asegurarlo, es posible que este individuo se pierda y no forme parte de la siguiente generación. Para evitar lo anterior se fuerza la selección de los mejores *n* individuos de la generación para que pasen intactos a la siguiente. A esta estrategia se le denomina elitismo y

puede ser generalizada especificando que permanezcan en la población los *n* mejores individuos de las pasadas *k* generaciones

❖ *Cruzamiento*

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

❖ *Mutación*

Ocasionalmente algunos elementos del código de ciertos individuos de un algoritmo genético se alteran a propósito. Éstos se seleccionan aleatoriamente en lo que constituye el símil de una mutación. El objetivo es generar nuevos individuos que exploren regiones del dominio del problema que probablemente no se han visitado aún. Esta exploración no presupone conocimiento alguno. Aleatoriamente se buscan nuevas soluciones posibles que quizá superen las encontradas hasta el momento. Esta es una de las características que hacen aplicables los algoritmos genéticos a gran variedad de problemas: no presuponer conocimiento previo acerca del problema a resolver ni de su dominio, no sólo en la mutación sino en el proceso total. De hecho, el problema a resolver sólo determina la función de evaluación y la manera de codificar las soluciones posibles. El resto de los subprocesos que constituyen el algoritmo son independientes y universalmente aplicables.

2.3. EL ALGORITMO GENÉTICO SIMPLE

El algoritmo genético simple, también denominado canónico, se representa en la Ilustración 2.

```
BEGIN /* Algoritmo Genetico Simple */
    Generar una poblacion inicial.
    Computar la funcion de evaluacion de cada individuo.
    WHILE NOT Terminado DO
    BEGIN /* Producir nueva generacion */
        FOR Tamaño poblacion/2 DO
        BEGIN /*Ciclo Reproductivo */
            Seleccionar dos individuos de la anterior generacion,
            para el cruce (probabilidad de seleccion proporcional
            a la funcion de evaluacion del individuo).
            Cruzar con cierta probabilidad los dos
            individuos obteniendo dos descendientes.
            Mutar los dos descendientes con cierta probabilidad.
            Computar la funcion de evaluacion de los dos
            descendientes mutados.
            Insertar los dos descendientes mutados en la nueva generacion.
        END
        IF  la poblacion ha convergido THEN
            Terminado := TRUE
    END
END
```

Ilustración 2: Pseudocógico del Algoritmo Genético Simple

Codificación del problema:

Se supone que los individuos (posibles soluciones del problema), pueden representarse como un conjunto de parámetros (que denominaremos genes), los cuales agrupados forman una ristra de valores (a menudo referida como cromosoma). Si bien el alfabeto utilizado para representar los individuos no debe necesariamente estar constituido por el {0, l}, buena parte de la teoría en la que se fundamentan los algoritmos genéticos utiliza dicho alfabeto.

Durante la fase reproductiva se seleccionan los individuos de la población para cruzarse y producir descendientes, que constituirán, una vez mutados, la siguiente generación de individuos. La selección de padres se efectúa al azar usando un procedimiento que favorezca a los individuos mejor adaptados, ya que a cada individuo se le asigna una probabilidad de ser seleccionado que es proporcional a su función de adaptación. Este procedimiento se dice que está basado en la ruleta sesgada. Según dicho esquema, los individuos bien adaptados se escogerán probablemente varias veces por generación, mientras que, los pobremente adaptados al problema, no se escogerán más que de vez en cuando.

Una vez seleccionados dos padres, sus cromosomas se combinan, utilizando habitualmente los operadores de cruce y mutación. Las formas básicas de dichos operadores se describen a continuación.

- El *operador de cruce* utilizado es el denominado cruzamiento en un punto y actúa del siguiente modo: coge dos padres seleccionados y corta sus ristras de cromosomas en una posición escogida al azar, para producir dos subristras iniciales y dos subristras finales. Después se intercambian las subristras finales, produciéndose dos nuevos cromosomas completos (véase la Ilustración 3). Ambos descendientes heredan genes de cada uno de los padres. Este operador se conoce como operador de cruce basado en un punto. Habitualmente el operador de cruce no se aplica a todos los pares de individuos que han sido seleccionados para emparejarse, sino que se aplica de manera aleatoria, normalmente con una probabilidad comprendida entre 0.5 y 1. En el caso en que el operador de cruce no se aplique, la descendencia se obtiene simplemente duplicando los padres.

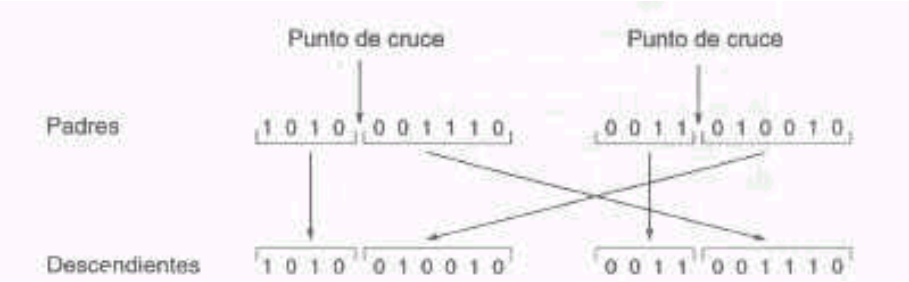

Ilustración 3: Operador de cruce basado en un punto

- El *operador de mutación* se aplica a cada hijo de manera individual, y consiste en la alteración aleatoria (normalmente con probabilidad pequeña) de cada gen componente del cromosoma. La Ilustración 4 muestra la mutación del quinto gen del cromosoma. Sí bien puede en principio pensarse que el operador de cruce es más importante que el operador de mutación, ya que proporciona una exploración rápida del espacio de búsqueda, éste último asegura que ningún punto del espacio de búsqueda tenga probabilidad cero de ser examinado, y es de capital importancia para asegurar la convergencia de los algoritmos genéticos.

Ilustración 4: Operador de mutación

Para criterios prácticos, es muy útil la definición de convergencia introducida en este campo por De Jong en su tesis doctoral en relación con los algoritmos genéticos simples. Si el algoritmo ha sido correctamente implementado, la población evolucionará a lo largo de las generaciones sucesivas de tal manera que la adaptación media extendida a todos los individuos de la población, así como la adaptación del mejor individuo se irán incrementando hacia el óptimo global. El concepto de convergencia está relacionado con la progresión hacia la uniformidad: un gen ha convergido cuando al menos el 95 % de los individuos de la población comparten el mismo valor para dicho gen. Se dice que la población converge cuando todos los genes han convergido. Se puede generalizar dicha definición al caso en que al menos un pequeño porcentaje de los individuos de la población hayan convergido.

La Ilustración 5 muestra como varía la adaptación media y la mejor adaptación en un algoritmo genético simple típico. A medida que el número de generaciones aumenta, es más probable que la adaptación media se aproxime a la del mejor individuo.

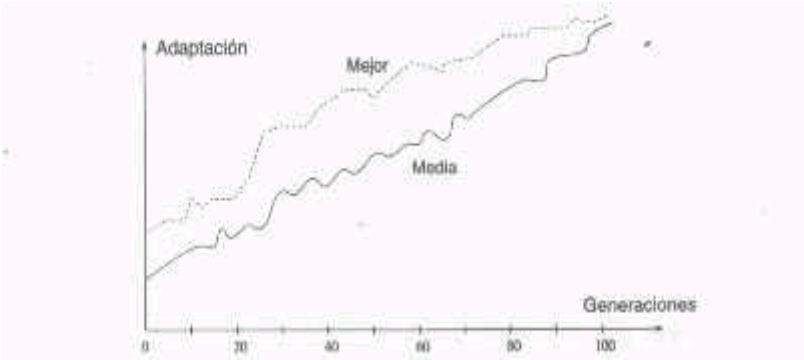

Ilustración 5: Adaptación media y mejor adaptación en un AG simple

Resumiendo, se puede explicar el proceso de un algoritmo genético simple del siguiente modo:

1. Decidir cómo codificar el dominio del problema.
2. Generar un conjunto aleatorio (población inicial) de N posibles soluciones codificadas al problema. A ésta se le llamará población actual.
3. Calificar cada posible solución (individuo) de la población actual.
4. Seleccionar dos individuos de la población actual con una probabilidad proporcional a su calificación.
5. Lanzar una moneda al aire (supongamos que la probabilidad de cara es p_c).
6. Si cayó cara mezclar los códigos de los dos individuos seleccionados para formar dos híbridos, a los que llamaremos nuevos individuos.
7. Si cayó cruz llamamos a los individuos seleccionados nuevos individuos.

8. Por cada bit de cada nuevo individuo lanzar otra moneda al aire (supongamos que en este caso cae cara con probabilidad p_m).
9. Si cae cara cambiar el bit en turno por su complemento.
10. Si cae cruz, el bit permanece inalterado.
11. Incluir a los dos nuevos individuos en una nueva población.
12. Si la nueva población tiene ya N individuos, llamarla población actural y regresar al paso 3, a menos que se cumpla alguna condición de terminación.
13. Si no, regresar al paso 4.

2.3.1. UN EJEMPLO SENCILLO DE ALGORITMO GENÉTICO SIMPLE

Como ilustración de los diferentes componentes del algoritmo genético simple, supongamos el problema adaptado de Goldberg de encontrar el máximo de la función $f(x) = x^2$ sobre los enteros {1,2,...,32}. Evidentemente para lograr dicho óptimo, bastaría actuar por búsqueda exhaustiva, dada la baja cardinalidad del espacio de búsqueda. Se trata por tanto de un mero ejemplo con el que pretendemos ilustrar el comportamiento del algoritmo anteriormente descrito. Consultando el pseudocódigo de la Figura 1, vemos que el primer paso a efectuar consiste en determinar el tamaño de la población inicial, para a continuación obtener dicha población al azar y computar la función de evaluación de cada uno de sus individuos.

Suponiendo que el alfabeto utilizado para codificar los individuos esté constituido por {0, 1}, necesitaremos ristras de longitud 5 para representar los 32 puntos del espacio de búsqueda.

En la Tabla 1, hemos representado los 4 individuos que constituyen la población inicial, junto con su función de adaptación al problema, así como la probabilidad de que cada uno de dichos individuos sea seleccionado según el modelo de ruleta sesgada para emparejarse (la probabilidad de que un individuo sea seleccionado es el cociente entre su función adaptación y la suma de las funciones de adaptación de sus competidores).

	Población inicial (fenotipos)	x valor genotipo	$f(x)$ valor (función adaptación)	$f(x)/\sum f(x)$ (probabilidad selección)	Probabilidad de selección acumulada
1	01101	13	169	0.14	0.14
2	11000	24	576	0.49	0.63
3	01000	8	64	0.06	0.69
4	10011	19	361	0.31	1.00
Suma			1170		
Media			293		
Mejor			576		

Tabla 1: Población inicial de la simulación efectuada a mano correspondiente al AG simple

Volviendo a consultar el pseudocódigo expresado en la Figura 1, vemos que el siguiente paso consiste en la selección de 2 parejas de individuos. Para ello es suficiente, con obtener 4 números reales provenientes de una distribución de probabilidad uniforme en el intervalo (0, 1), y compararlos con la última columna de la Tabla l. Así por ejemplo, supongamos que dichos 4 números hayan sido: 0.58; 0.84; 0.11 y 0.43. Esto significa que los individuos seleccionados para el cruce han sido: el individuo 2 junto con el individuo 4, así como el individuo 1 junto con el individuo 2 (para ello basta fijarnos en que un individuo i se elige cuando el número obtenido al azar está comprendido entre la probabilidad de selección acumulada del individuo i-1 y la probabilidad de selección acumulada del individuo i.

Para seguir con el algoritmo genético simple, necesitamos determinar la probabilidad de cruce (p). Supongamos que se fije en p = 0.8. Valiéndonos al igual que antes de números provenientes de la distribución uniforme (dos en este caso), determinaremos si los emparejamientos anteriores se llevan a cabo.

Emparejamiento de los individuos seleccionados	Punto de cruce	Descendientes	Nueva población descendientes mutados	x valor genotipo	$f(x)$ función adaptación
11000	2	11011	11011	27	729
10011	2	10000	10000	16	256
01101	3	01100	11100	28	784
11000	3	11001	11101	29	841
Suma					2610
Media					652.5
Mejor					841

Tabla 2: Población en el tiempo 3, proveniente de efectuar los operadores de cruce y mutación sobre los individuos de la Tabla 1

Admitamos, por ejemplo, que los dos números extraídos sean menores que 0.8, decidiéndose por tanto efectuar el cruce entre las dos parejas. Para ello escogeremos un número al azar entre 1 y L (siendo L la longitud de la ristra utilizada para representar el individuo). Notése que la restricción impuesta al escoger el número entre 1 y L, se realiza con la finalidad de que los descendientes no coincidan con los padres.

Supongamos, tal y como se indica en la Tabla 2, que los puntos de cruce resulten ser 2 y 3. De esta manera obtendríamos los 4 descendientes descritos en la tercera columna de la Tabla 2. A continuación siguiendo el pseudocódigo de la Figura 1, mutaríamos con una probabilidad p, cercana a cero, cada uno de los bit de las cuatro ristras de individuos. En este caso suponemos que el único bit mutado corresponde al primer gen del tercer individuo. En las dos últimas columnas se pueden consultar los valores de los individuos, así como las funciones de adaptación correspondientes. Como puede observarse, tanto el mejor individuo como la función de adaptación media han mejorado sustancialmente al compararlos con los resultados de la Tabla 1.

2.4. EL ALGORITMO GENÉTICO PARALELO

Un programa es paralelo si en cualquier momento de su ejecución puede ejecutar más de un proceso. Para crear programas paralelos eficientes hay que poder crear, destruir y especificar procesos así como la interacción entre ellos. Básicamente existen tres formas de paralelizar un programa:

- Paralelización de grano fino: la paralelización del programa se realiza a nivel de instrucción. Cada procesador hace una parte de cada paso del algoritmo (selección, cruce y mutación) sobre la población común.

- Paralelización de grano medio: los programas se paralelizan a nivel de bucle. Esta paralelización se realiza habitualmente de una forma automática en los compiladores.

- Paralelización de grano grueso: se basan en la descomposición del dominio de datos entre los procesadores, siendo cada uno de ellos el responsable de realizar los cálculos sobre sus datos locales.

La computación paralela se ha convertido en una parte fundamental en todas las áreas de cálculo científico, ya que permite la mejora del rendimiento simplemente con la utilización de un mayor número de procesadores, memorias y la inclusión de elementos de comunicación que permitan a los procesadores trabajar conjuntamente para resolver un determinado problema.

En el caso que nos ocupa, podemos decir que los algoritmos genéticos tienen una estructura que se adapta perfectamente a la paralelización. De hecho la evolución natural es en si un proceso paralelo ya que evoluciona utilizando varios individuos. Los principales métodos de paralelización de AGs consisten en la división de la población en varias subpoblaciones.

El tamaño y distribución de la población entre los distintos procesadores será uno de los factores fundamentales a la hora de paralelizar el algoritmo.

Existen varias formas de paralelizar un algoritmo genético. La primera y más intuitiva es la global, que consiste básicamente en paralelizar la evaluación de los individuos manteniendo una población. Otra forma de paralelización global consiste en realizar una ejecución de distintos AGs secuenciales simultáneamente (estas dos formas de paralelización no cambian la estructura del algoritmo utilizado). El resto de aproximaciones sí cambian la estructura del algoritmo y dividen la población en subpoblaciones que evolucionan por separado e intercambian individuos cada cierto número de generaciones. Si las poblaciones son pocas y grandes, tenemos la paralelización de grano grueso. Si el número de poblaciones es grande y con pocos individuos en cada población tenemos la paralelización de grano fino. Por ultimo, existen algoritmos que mezclan propiedades de estos dos últimos y que se denominan mixtos.

Además de conseguir tiempos de ejecución menores, al paralelizar un AG estamos modificando el comportamiento algorítmico, y esto hace que podamos obtener otras soluciones y experimentar con las distintas posibilidades de implementación y los distintos factores que influyen en ella. Estas poblaciones van evolucionando por separado para detenerse en un momento determinado e intercambiar los mejores individuos entre ellas. Técnicamente hay 3 características importantes que influyen en la eficiencia de un algoritmo genético paralelo:

- La topología que define la comunicación entre subpoblaciones.
- La proporción de intercambio: número de individuos a intercambiar.
- Los intervalos de migración: periodicidad con que se intercambian los individuos.

2.4.1. MODELOS DE ISLAS

En este apartado se introducirán tres maneras diferentes de explotar el paralelismo de los Algoritmos Genéticos, por medio de los denominados modelos de islas.

La idea básica consiste en dividir la población total en varias subpoblaciones en cada una de las cuales se lleva, a cabo un Algoritmo Genético. Cada cierto número de generaciones, se efectúa un intercambio de información entre las subpoblaciones, proceso que se denomina migración. La introducción de la migración hace que los modelos de islas sean capaces de explotar las diferencias entre las diversas subpoblaciones, obteniéndose de esta manera una fuente de diversidad genética. Cada subpopulación es una "isla", definiéndose un procedimiento por medio del cual se mueve el material genético de una "isla" a otra. La determinación de la tasa de migración, es un asunto de capital importancia, ya que de ella puede depender la convergencia prematura de la búsqueda.

Se pueden distinguir diferentes modelos de islas en función de la comunicación entre las subpoblaciones. Algunas comunicaciones típicas son las siguientes:

- Comunicación en estrella, en la cual existe una subpoblación que es seleccionada como maestra (aquella que tiene mejor media en el valor de la función objetivo), siendo las demás consideradas como esclavas. Todas las subpoblaciones esclavas mandan sus h1 mejores individuos (h1 > 1) a la subploblación maestra la cual a su vez manda sus h2 mejores individuos (h2 >1) a cada una de las subpoblaciones esclavas.

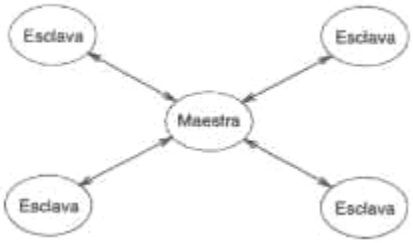

Ilustración 6: AG paralelo. Modelo de islas. Comunicación en estrella

- Comunicación en red, en la cual no existe una jerarquía entre las subpoblaciones, mandando todas y cada una de ellas sus h3 (h3 > 1) mejores individuos al resto de las subpoblaciones.

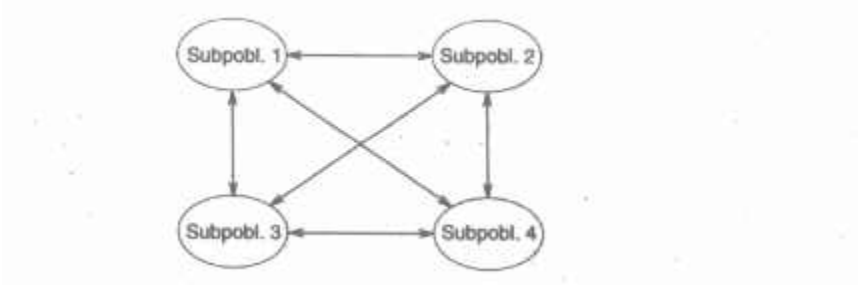

Ilustración 7: AG paralelo. Modelo de Islas. Comunicación en red

- Comunicación en anillo, en la cual cada subpoblación envía sus h4 mejores individuos (h4 > 1), a una población vecina, efectuándose la migración en un único sentido de flujo.

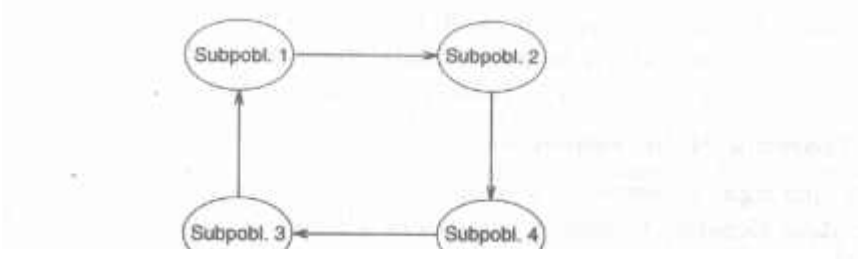

Ilustración 8: AG paralelo. Modelo de Islas. Comunicación en anillo

Se comentará algo más sobre algoritmos genéticos paralelos cuando se desarrolle el comportamiento de los algoritmos genéticos en el problema del agente viajero.

2.5. ¿POR QUÉ FUNCIONA UN ALGORITMO GENÉTICO?

Uno de los argumentos básicos de la teoría de la evolución es que los individuos que presentan semejanzas están emparentados. Basándose en este principio, Holland observó que ciertos conjuntos de individuos con determinadas similitudes en algunas posiciones de sus cadenas poseían propiedades comunes buenas y otros por el contrario eran peores.

Por ejemplo en la función $f(x) = x^2$, si codificamos en cadenas de cuatro bits, los que empiezan con un 1 tienen imágenes de valor mayor. Es decir, el valor medio sobre las cadenas de la forma 1*** está por encima de la media global de la población. Abstrayendo esta idea, Holland define el concepto de *esquema* (H) en una codificación binaria con cadenas de longitud L como

$$H = h_{L-1} \ldots h_0 \in \{0, 1, *\}^L \leftrightarrow H = \{ s_{L-1} \ldots s_0 / h_j \neq * \rightarrow s_j = h_j \} .$$

$$\mathbf{E}(n(H, t+1)) = n(H, t) \cdot N \cdot \frac{f(H, t)}{\sum_{s \in P(t)} f(s)}$$

Es decir, un esquema representa un cierto subconjunto de la población $\{0, 1\}^L$ en el que sus individuos se diferencian, a lo sumo, en las posiciones de los asteriscos. Por ejemplo, el esquema H = 10 * 01* se corresponde con el conjunto de cadenas

$$\{\underline{10}0\underline{01}0, \underline{10}0\underline{01}1, \underline{10}1\underline{01}0, \underline{10}1\underline{01}1\}$$

Por otra parte, cualquier conjunto C de cadenas define un esquema, basta considerar la proyección sobre la j-ésima coordenada

$$\pi j : \begin{array}{ccc} \{0, 1\} & \longrightarrow & \{0, 1\} \\ s_{L-1} \ldots s_0 & \longmapsto & s_j \end{array}$$

y definir el esquema H como:

$$hj = \begin{cases} 0 & si\ \pi_j(H) = \{0\} \\ 1 & si\ \pi_j(H) = \{1\} \\ * & si\ \pi_j(H) = \{0\ ,\ 1\} \end{cases}$$

De hecho, el conjunto de cadenas que se pueden generar por cruces de elementos del conjunto C es exactamente H. Por ejemplo, si C = {001011, 011111} entonces cada cadena de H = 0 * 1 * 11 se puede generar cruzando los elementos de C, incluso 011011 y 001111, que no estaban inicialmente en C.

Obviamente en unos esquemas sus elementos guardan más parecido entre sí que en otros. Para cuantificar esta idea existen dos conceptos: el *orden* de un esquema que es el número de alelos fijos en el esquema y la *longitud de definición* que es la distancia entre el primero y el último alelo fijos. Por ejemplo, si $H = 00 * * 1*$, entonces $o(H) = 3$ y $\delta(H) = 4$.

Claramente, a mayor longitud de definición (o a mayor orden) del esquema, más probabilidad de sufrir un cruce o una mutación que destruya la estructura del mismo. Parece pues que los esquemas cortos, de orden bajo son más proclibes a mantenerse en la población y si estos esquemas tienen representantes en la población con idoneidad alta, estas cadenas serán seleccionadas con mayor frecuencia y generarán más representantes en la generación siguiente. Esto en grosso modo es lo que dice el *Teorema de los Esquemas* que Holland demostró en 1975 y que es el resultado teórico más importante que existe en este campo. En esencia, este teorema afirma que el algoritmo dirige la búsqueda del óptimo a través de ciertos subconjuntos de la población; es decir, explora el espacio de búsqueda a través de aquellas áreas que, en media, son más adecuadas.

Dado que durante la reproducción un cromosoma es seleccionado con una probabilidad proporcional a su idoneidad

$$\frac{f(s)}{\sum_{s \in P(t)} f(s)}$$

donde $P(t)$ denota la población en la generación t-ésima, entonces si partimos de una población de N elementos, el número esperado de representantes de H en la iteración siguiente es:

$$\mathbf{E}(n(H,t+1)) = n(H,t) \cdot N \cdot \frac{f(H,t)}{\sum_{s \in P(t)} f(s)}$$

donde **E** denota el operador esperanza, $n(H, t)$ es el número de cadenas del esquema en la generación t-ésima y $f(H,t) = \frac{\sum_{s \in H} f(s)}{n(H,t)}$ es el valor medio del esquema en esa generación. Si ahora consideramos la acción de los operadores de cruce y mutación, la anterior ecuación se transforma en:

$$\mathbf{E}(n(H,t+1)) = n(H,t) \cdot \frac{f(H,t)}{\overline{f}} [1 - \alpha(H)]$$

donde $\overline{f} = \frac{\sum_{s \in P(t)} f(s)}{N}$ es la idoneidad media de la población y $\alpha(H)$ depende de la estructura de H y de las probabilidades de cruce y mutación, *pc* y *pm* respectivamente, pero no de t, pues

$$\alpha(H) = p_c \cdot \frac{\delta(H)}{\ell - 1} \cdot (1 - p_m)^{o(H)}$$

Esta expresión, al despreciar los términos de grado ≥ 2 en el binomio de Newton, y como consecuencia de que en la mayoría de los casos $p_m \approx 0.01$, se convierte en la formulación definitiva del *Teorema Fundamental de los algoritmos genéticos (o Teorema de los esquemas):*

$$\mathbf{E}(n(H, t+1)) \geq n(H, t) \cdot \frac{f(H, t)}{\overline{f}} \left[1 - p_c \cdot \frac{\delta(H)}{\ell - 1} - o(H) \cdot p_m\right]$$

De modo que que si *H* es un esquema con una idoneidad superior a la media de la población, se espera que se incremente el número de cadenas con la estructura de *H* en la generación siguiente, siempre que α sea pequeña. Es decir, la tesis de este teorema se puede interpretar diciendo que *"esquemas cortos, de orden bajo, con idoneidad sobre la media, incrementan el número de representantes en generaciones sucesivas".* Este tipo de esquemas –que parecen jugar un papel importante en la actuación de los AGs–, se conocen como bloques de construcción o *building blocks*.

Parece pues que yuxtaponiendo bloques sólidos de tamaño pequeño se pueden llegar a construir individuos cada vez mejores. Esto llevó a pensar durante algún tiempo que funciones que se pudiesen definir utilizando esquemas cortos, de orden bajo y altamente idóneos serían fáciles de optimizar por los AGs. Esta afirmación –conocida como *la hipótesis de los bloques constructivos* – parece muy razonable; de hecho, se han diseñado AGs para aplicaciones variadas que son una evidencia empírica de que, para diferentes tipos de problemas, dicha hipótesis es correcta. Sin embargo, en 1993 Forrest y Mitchell definen dos funciones (*Royal Road functions*) que la contradicen. La primera de ellas, R_1, se define a partir de los esquemas:

$$H1 = 1^{(8)} * {}^{(56)}$$
$$H2 = *{}^{(8)}\, 1^{(8)} * {}^{(48)}$$
$$H3 = *{}^{(16)}\, 1^{(8)} * {}^{(40)}$$
$$H4 = *{}^{(24)}\, 1^{(8)} * {}^{(32)}$$
$$H5 = *{}^{(32)}\, 1^{(8)} * {}^{(24)}$$
$$H6 = *{}^{(40)}\, 1^{(8)} * {}^{(16)}$$
$$H7 = *{}^{(48)}\, 1^{(8)} * {}^{(8)}$$
$$H8 = *{}^{(56)}\, 1^{(8)}$$

donde $\alpha^{(n)}$ denota la cadena $\overbrace{\alpha\alpha\ldots\alpha}^{n}$ y $R_1 : \{0, 1\}^{64} \rightarrow \Re$, viene dada por $R_1(s) = \sum_{i:s\in H_i} 8$, para cada $s \in \{0, 1\}^{64}$. Por ejemplo $R_1(1^{(8)}, 0^{(56)}) = 8$

mientras que $R_1(1^{(16)}, 0^{(48)}) = 16$. El máximo de esta función se alcanza en la cadena s = 11. . .11 = $1^{(64)}$.

La segunda función Royal Road, R_2, se define de forma similar a la primera, a partir de los esquemas anteriores añadiendo los siguientes:

$$H9 = 1^{(16)} * ^{(48)}$$
$$H10 = * ^{(16)} 1^{(16)} * ^{(32)}$$
$$H11 = * ^{(32)} 1^{(16)} * ^{(16)}$$
$$H12 = * ^{(48)} 1^{(16)}$$
$$H13 = 1^{(32)} * ^{(32)}$$
$$H14 = * ^{(32)} 1^{(32)}$$

y con $R_2(s) = \sum_{i:s\in H_i} c_i$, donde, como antes, $c_i = 8$ para $1 \leq i \leq 8$ y $c_i = 16$ (respectivamente $c_i = 32$) para $9 \leq i \leq 12$ (resp. $13 \leq i \leq 14$). Obviamente, esta función alcanza el máximo en la misma cadena que la anterior función.

Es razonable esperar que estas dos funciones sean fáciles para los AGs, porque están definidas por medio de esquemas cortos, de orden bajo, que se pueden utilizar como un camino directo (*Royal Road*) hacia la solución óptima. Pero, además, la función R_2 debería ser más fácil que R_1 ya que se utilizan más esquemas en la definición.

Como medida de la dificultad, que el algoritmo encuentra para optimizar funciones, se puede considerar el número medio de generaciones necesarias para que se cumplan las tres condiciones siguientes:

- al menos un elemento de la población tenga valor máximo,

- la idoneidad media de la población sea superior al 90% del valor máximo (en los casos en los que no se conoce el valor del máximo, se exige que un porcentaje alto de la población alcance un valor umbral prefijado),

- la desviación típica tenga un valor menor o igual que el 5% de la media.

Se ha demostrado que estas funciones son difíciles para los AGs y de hecho, comparando el comportamiento de ambas, resulta que R_2 es realmente más difícil que R_1. Esto no contradice el teorema de los esquemas puesto que la explicación del comportamiento de los algoritmos genéticos mediante el citado teorema es incompleta; no precisa cómo se comporta el algoritmo frente a

esquemas sin representantes en la población y, en consecuencia, no informa de la dirección de búsqueda en las iteraciones sucesivas. De hecho, las medias $f(H, t)$ son medias de esquemas presentes en la población actual y, por tanto, el algoritmo carece de referencias a nuevos esquemas que se introduzcan como consecuencia de cruces o mutaciones. Tampoco se informa sobre lo que sucede si no hay "buenos" esquemas en la población de partida y se debe esperar a que eventuales cruces o mutaciones los generen. Mientras ello sucede el algoritmo carece de información que le guíe a zonas donde estén las cadenas mejores. Eso puede hacer que emplee demasiado tiempo en la búsqueda del óptimo y que no resulte eficiente. Por tanto, hay funciones más fáciles que otras a la hora de ser optimizadas mediante el uso de los algoritmos genéticos.

3. ALGUNAS APLICACIONES DE LOS ALGORITMOS GENÉTICOS

En esta sección se hablará de algunas de las aplicaciones de los algoritmos genéticos en diversos temas matemáticos.

3.1. UNA APLICACIÓN DE LOS ALGORITMOS GENÉTICOS EN LA DISCRIMINACIÓN

El análisis discriminante forma parte del conjunto de técnicas estadísticas diseñadas para resolver el problema de clasificación. Se caracteriza por estudiar la relación entre una variable categórica dependiente (el grupo de clasificación) y un conjunto de variables reales (posiblemente vectoriales) independientes. Estas describen a cada uno de los individuos a clasificar. A través de esta técnica, se construyen reglas de decisión que permiten discriminar o separar grupos mediante funciones de las variables observadas**,** minimizando la probabilidad de clasificación errónea, o bien maximizándola si la clasificación es correcta. Las reglas más usuales se construyen a partir de un modelo probabilístico, y de muestras de entrenamiento.

Presentaremos una aplicación de los algoritmos genéticos a las funciones discriminantes que conforman la población inicial, las cuales se generan realizando el análisis discriminante de cada muestra obtenida por la técnica de remuestreo que se aplicó a una muestra de entrenamiento. Posteriormente, las funciones se evalúan para seleccionar las mejores y, a través de la aplicación de operadores genéticos, llegar a obtener una o un grupo de funciones que discriminen mejor que la proporcionada por el método de Fisher.

Para comprender adecuadamente el desarrollo de la aplicación de la técnica propuesta, se describirá en qué consiste el análisis discriminante.

3.1.1. EL ANÁLISIS DISCRIMINANTE

Bajo el supuesto de ***g*** grupos o poblaciones Π_i **,** i = 1,. .. , g mutuamente excluyentes, con probabilidades a priori asociadas a cada población como $\alpha_i \geq O$ y $\Sigma \alpha_i = 1$ y vector de variables aleatorias $X = (X,, ..., Xp)$, el objetivo que se persigue es obtener un conjunto de $g (g - 1)/2$ funciones lineales o pseudolineales del vector X tal que maximice la separación entre los grupos y permita minimizar la probabilidad de clasificación errónea en cada grupo. Bajo los supuestos de que $g = 2$ y de que la separación entre los grupos es lineal o pseudolineal, una regla de decisión para este caso viene dada por:

La unidad *u* con vector de variables observadas x clasifica en Π_1
si $a'x \geq 0$

La unidad *u* con vector de variables observadas x clasifica en Π_2
si $a'x \leq 0$

siendo $a \in \Re^{p+1}$ tal que minimice la probabilidad de clasificación errónea, es decir,

$$a^* = \min_{a \in \Re^{P+1}} (\alpha_2 P (a' x \leq 0 / x \in \Pi_2) + \alpha_1 P (a' x \geq 0 / x \in \Pi_1)$$

Dada una muestra de entrenamiento X de tamaño N x p con N = Σn_j j = 1, 2 y la regla de clasificación anterior, se construye la regla de clasificación empírica a partir de la función discriminante estimada a'(X) x al ser a'(X) una estimación del vector de parámetros a tal que minimice la estimación de la probabilidad de clasificación errónea condicionada a la muestra de entrenamiento, es decir, la selección del vector a debe cumplir:

$$(a^* / X) = \min_{a \in \Re^{P+1}} (\alpha_2 P(a'(X) x \leq 0 / x \in \Pi_2, X)+\alpha_1 P(a'(X) x \geq 0 /x \in \Pi_1, X)$$

con $P (a' x \leq 0 / x \in \Pi_2, X) = [\# (x \in \Pi_2 / a'(X) x \leq 0)] / n_2$
$P (a' x \geq 0 / x \in \Pi_1, X) = [\# (x \in \Pi_1 / a'(X) x \geq 0)] / n_1$

Ahora veremos qué tipo de selección y de operadores genéticos vamos a utilizar para resolver problemas asociados al análisis discriminante.

- *Selección de mejores cromosomas:*
  La idea es escoger los mejores cromosomas o soluciones en una población que se va a reproducir para formar generaciones sucesivas; esta selección se realiza aleatoriamente, y se asigna una probabilidad P_j a cada elemento j , perteneciente a la población en cuestión y se toma como base el valor de ajuste. También se generan números aleatorios con

distribución uniforme, uno para cada elemento de la población, y se comparan contra la probabilidad acumulada

$c_i = \sum_{j=1}^{n} P_j$. Entonces, la solución o cromosoma i se selecciona para integrar la

nueva población si $c_{i-1} < U(0, 1) < c_i$

- *Operadores genéticos: cruzamiento y mutación:*
  En el problema que se presenta se utilizaron los siguientes operadores:
  Cruzamiento aritmético: sean S_m, y S_k, dos soluciones o cromosomas, entonces al seleccionar aleatoriamente un valor ***r*** de una variable aleatoria con distribución $U(0, l)$, se logra la generación de dos nuevas soluciones o nuevos cromosomas al considerar las combinaciones convexas entre los dos, es decir:

  $$S'_m = r S_m + (1 - r) S_k \quad \text{y} \quad S'_k = r S_k + (1 - r) S_m$$

  De esta forma se generan dos nuevas soluciones.
  Mutación uniforme: sea $S(i)$ la matriz de tamaño s x p cuyas filas representan s posibles soluciones del problema en la iteración i con p componentes, y

  $$a_j = \min_i S_{i,j} \quad \text{y} \quad b_j = \max_i S_{i,j} \qquad \text{donde } j = 1, \ldots, p$$

  es decir, el límite inferior y superior del conjunto de cada columna de la matriz $S(i)$ del problema. Seleccionando aleatoriamente el valor k entre 1 y p y un valor de la variable aleatoria $U(a_k, b_k)$ con distribución uniforme en el intervalo (a_k, b_k) , se realiza la siguiente mutación:
  $S_{i,j} = U(a_k, b_k) \quad i = 1,\ldots,p \quad si \quad j = k$
  $S_{i,j} = S_{i,j} \quad i = 1,\ldots,p \quad si \quad j \neq k$

3.1.2. UNA APLICACIÓN PRÁCTICA DE LOS ALGORITMOS GENÉTICOS EN EL ANÁLISIS DISCRIMINANTE

Para ilustrar el comportamiento de esta técnica, se usó la base de datos de IRIS dada por Fisher (1936). Se consideran dos especies de flores (Versicolor y Virginica), con un total de 50 casos por cada especie y cuatro variables:
X1 =longitud del sépalo
X2 = ancho del sépalo
X3 = longitud del pétalo
X4 = ancho del pétalo.

Los cálculos del procedimiento propuesto anteriormente, se realizaron con el apoyo de los paquetes SPLUS 2000 y MATLAB 12. La solución obtenida bajo los supuestos de normalidad y de homogenidad de las matrices de covarianzas de las dos especies, y considerando la muestra de entrenamiento de tamaño *100,* se obtuvo como vector *a'(X)* de coeficentes de la función discriminante:
(3.5563, 5.5786, -6.97013, -l2.386, 16.663)
con una estimación del error de clasificación de 0.03.

Después de aplicar las técnicas de remuestreo a la muestra original, se generaron 12 muestras aleatorias, a las cuales se les aplicó la discriminación bajo los mismos supuestos. Las soluciones dadas del vector *a'(X)* para cada muestra aparecen en el Tabla 3, que muestran los cromosomas progenitores o población inicial. La última columna del cuadro indica el error estimado, resultante de reclasificar la muestra de entrenamiento original con las 12 soluciones resultantes.

Sepal	Sepaw	Petal	Petaw	Constant	Error
4.5665	7.2112	-8.3568	-14.0763	15.4254	.04
0.10279	7.49824	-3.5749	-14.9522	20.8127	.06
1.8046	6.21916	-4.3873	-16.1442	19.7121	.02
2.7065	6.2569	-6.1654	-12.889	17.7004	.038
4.5665	7.2112	-8.3568	-14.0763	15.4254	.04
3.556	5.5786	-6.97013	-12.386	16.663	.03
2.6769	8.234	-7.45	-16.3613	24.5204	.03
3.7958	8.027	-6.823	-16.5313	14.639	.046
4.7955	4.2296	-7.2728	-14.0795	16.89312	.02
1.8046	6.2192	-4.38726	-16.1442	19.7112	.02
6.6358	5.4506	-10.48465	-12.41914	14.9495	.01
6.3973	4.13926	-9.04369	-13.71768	15.15608	.00

Tabla 3: Cromosomas progenitores

Asignando una probabilidad de frecuencia 1 a cada error (es decir, la probabilidad de cada error será 1 / 12) y al generar 12 números aleatorios con distribución *U(0, l),* ordenados de menor a mayor, se procede a utilizar la técnica de selección. Las Tablas 4 y 5 muestran los resultados de la primera selección.

Error	Prob.Acum.	$(c_{i-1} < U(0,1) < c_i)$	
.04	.083	(0 <.01 <.083)	Si
.06	.166	(.083 <.23 <.166)	No
.02	.249	(.166 <.44 <.249)	No
.038	.332	(.249 <.45 <.332)	No
.04	.415	(.332 <.48 <.415)	No
.03	.498	(.415 <.60 <.498)	No
.03	.581	(.498 <.61 <.581)	No
.046	.664	(.581 <.76 <.664)	No
.02	.747	(.664 <.79 <.747)	No
.02	.83	(.747 <.82 <.83)	Si
.01	.913	(.83 <.89 <.913)	Si
.00	1	(.913 <.95 <1)	Si

Tabla 4: Primera Selección de Cromosomas Progenitores

Sepal	Sepaw	Petal	Petaw	Constant	Error
4.5665	7.2112	-8.3568	-14.0763	15.4254	.04
1.8046	6.2192	-4.38726	-16.1442	19.7112	.02
6.6358	5.4506	-10.48465	-12.41914	14.9495	.01
6.3973	4.13926	-9.04369	-13.71768	15.15608	.00

Tabla 5: Combinaciones seleccionadas

A continuación se realizó el primer cruce aritmético (Tabla 6) y se obtuvieron las nuevas combinaciones o nuevas soluciones.

Sepal	Sepaw	Petal	Petaw	Constant	Error
3.07507	6.67555	-6.213248	-15.192968	17.740184	.02
3.29602	6.7548	-6.5308	-15.0275	17.397266	.02
5.3801	6.5189	-9.1935	-13.4247	15.2383	.03
5.8222	6.1429	-9.648	-13.0707	15.1366	.03
5.5075	5.6322	-8.7099	-13.8919	15.2870	.03
5.4563	5.7182	-8.6906	-13.902	15.2945	.03
2.3298	6.1357	-5.05	-15.7393	19.1944	.02
6.1106	5.5341	-9.8219	-12.8241	15.4672	.03
2.8972	5.7244	-5.4950	-15.5669	18.6282	.03
5.3047	4.6341	-7.9359	-14.2949	16.24	.03
6.5062	4.7379	-9.7015	-13.1249	15.0618	.03
6.5269	4.8520	-9.8269	-13.0119	15.0438	.03

Tabla 6: Resultado del primer cruce aritmético

Para realizar la primera mutación uniforme, se seleccionó un número aleatorio entre 1 y 5, en este caso resultó $j = 4$ y se generó un número aleatorio (gene) entre los límites superior e inferior del vector de soluciones para la columna 4. El valor obtenido fue 13.2639, el cual se sustituyó en la columna correspondiente para cada uno de los cromosomas. En la Tabla 7 se dan los resultados obtenidos.

Sepa1	Sepaw	Petal	Petaw	Constant	Error
3.07507	6.67555	-6.213248	-13.2639	17.740184	.12
3.29602	6.7548	-6.5308	-13.2639	17.397266	.08
5.3801	6.5189	-9.1935	-13.2639	15.2383	.03
5.8222	6.1429	-9.648	-13.2639	15.1366	.02
5.5075	5.6322	-8.7099	-13.2639	15.2870	.03
5.4563	5.7182	-8.6906	-13.2639	15.2945	.03
2.3298	6.1357	-5.05	-13.2639	19.1944	.15
6.1106	5.5341	-9.8219	-13.2639	15.4672	.03
2.8972	5.7244	-5.4950	-13.2639	18.6282	.14
5.3047	4.6341	-7.9359	-13.2639	16.24	.05
6.5062	4.7379	-9.7015	-13.2639	15.0618	.02
6.5269	4.8520	-9.8269	-13.2639	15.0438	.02

Tabla 7: Primera mutación uniforme

Las Tablas 8, 9 y 10 muestran los resultados del segundo cruce. Para realizar la segunda mutación se seleccionó un número aleatorio entre 1 y 5, sin considerar el 4, que ya corresponde a la columna mutada anteriormente. En este caso, la nueva columna a mutar es j = 5 y el valor aleatorio elegido entre 15.0474 y 15.2138 es 15.1394. En la Tabla 11 se dan los resultados.

Error	Prob.Acum.	$(c_i - 1 < U(0,1) < c_i)$	
.12	.083	(0<.151<.083)	No
.08	.166	(.083 < .235 < .166)	No
.03	.249	(.166 < .240 < .249)	Si
.02	.332	(.249 < .334 < .332)	No
.03	.415	(.332 < .591< .415)	No
.03	.498	(.415<.718<.498)	No
.15	.581	(.498 < .738 < .581)	No
.03	.664	(.581<.792<.664)	No
.14	.747	(.664 < .875 < .747)	No
.05	.83	(.747 < .894 < .83)	No
.02	.913	(.83 < .897 < .913)	Si
.02	1	(.913 < .939 <1)	Si

Tabla 8: Segunda selección

Sepal	Sepaw	Petal	Petaw	Constant	Error
5.3801	6.5189	-9.1935	-13.2639	15.2383	.03
6.5062	4.7379	-9.7015	-13.2639	15.0618	.02
6.5269	4.8520	-9.8269	-13.2639	15.0438	.02

Tabla 9: Combinaciones seleccionadas

Sepal	Sepaw	Petal	Petaw	Constant	Error
6.3498	4.9853	-9.6309	-13.2639	15.0863	.03
5.5365	6.2715	-9.2641	-13.2639	15.2138	.03
6.2943	5.19	-9.6984	-13.2639	15.0832	.02
5.6127	6.1809	-9.3220	-13.2639	15.1989	.03
6.5228	4.8293	-9.8020	-13.2639	15.0474	.02
6.5103	4.7606	-9.7264	-13.2639	15.0582	.02

Tabla 10: Segundo Cruce Aritmético

Sepal	Sepaw	Petal	Petaw	Constant	Error
6.3498	4.9853	-9.6309	-13.2639	15.1394	.03
5.5365	6.2715	-9.2641	-13.2639	15.1394	.03
6.2943	5.19	-9.6984	-13.2639	15.1394	.02
5.6127	6.1809	-9.3220	-13.2639	15.1394	.03
6.5228	4.8293	-9.8020	-13.2639	15.1394	.02
6.5103	4.7606	-9.7264	-13.2639	15.1394	.02

Tabla 11: Segunda Mutación

Continuando este procedimiento Puede observarse como la estimación del error de clasificación va estabilizándose a 0.02. Una solución subópbtima del problema de discriminación original viene dada por la función discriminante lineal:

6.3981 X1 + 5.0345 X2 - 9.7445 X3 - 13.2639 X4 +15.1394

con una estimación del error de clasificación de 0.02.

En este ejemplo se ha mostrado que la técnica de algoritmos geneticos puede usarse para refinar los métodos clásicos de discriminación. Esta afirmación es soportada por el hecho de que el método de clasificación de Fisher aplicado al conjunto de datos de IRIS, tiene un error de clasificacion de 0.03; al aplicar algoritmos genéticos el método se mejora al obtenerse un error de clasificación de 0 .02 o menor.

3.2. UNA APLICACIÓN DE LOS ALGORITMOS GENÉTICOS AL PROBLEMA DEL AGENTE VIAJERO

El problema del agente viajero, también denominado TSP (Travelling Salesman Problem), consiste en, dada una colección de ciudades, determinar el recorrido de coste mínimo, visitando cada ciudad exactamente una vez y volviendo al punto de partida.

Más precisamente, dado un número entero $n \geq 3$ y dada una matriz $C = (cij)\ \epsilon\ M(n, n)$, con elementos cij enteros no negativos, se trata de encontrar la permutación cíclica Π de los enteros de 1 a n que minimiza

$$\sum_{i=1}^{n} c_{\Pi(i),\Pi(i+1)} \quad \text{con } \Pi(n+1)=1$$

A lo largo de los años el problema del agente viajero ha ocupado la mente de numerosos investigadores. Los motivos son varios. En primer lugar, el TSP es un problema muy sencillo de enunciar, pero muy difícil de resolver. En segundo lugar, el TSP es aplicable a una gran variedad de problemas de planificación. Finalmente, se ha convertido en una especie de problema test, es decir los nuevos métodos de optimización combinatoria son a menudo aplicados al TSP con objeto de tener una idea de sus potencialidades.

La primera aproximación al TSP a partir de Algoritmos Genéticos la efectuó Brady (1985). Su intento fue seguido por Grefenstette y col. (1985), Goldberg y Lingle (1985), Oliver y col. (1987) y otros muchos.

Se expondrán algunas representaciones, así como diferentes operadores de cruce y mutación, que han sido utilizados para resolver el TSP por medio de Algoritmos Genéticos. En particular veremos la representación basada en la trayectoria que es la más utilizada, y a continuación se explicará la representación binaria que presenta algunos problemas a la hora de realizar el cruzamiento y mutación clásicos y, por ese motivo, tuvo que idearse un método distinto para resolver el TSP utilizando este tipo de representación.

3.2.1. REPRESENTACIÓN BASADA EN LA TRAYECTORIA

La representación basada en la trayectoria es la representación más natural de una gira (o recorrido). En ella, una gira se representa como una lista de n ciudades. Si la ciudad i es el j-ésimo elemento de la lista, la ciudad i es la j-ésima ciudad a visitar. Así por ejemplo, la gira $3 - 2 - 4 - 1 - 7 - 5 - 8 - 6$ se representará como:

(3 2 4 1 7 5 8 6).

Debido a que los operadores clásicos no funcionan con esta representación, se han definido otros operadores de cruce y mutación, algunos de los cuales se describen a continuación.

3.2.1.A) Operadores de cruce

I. OPERADOR DE CRUCE BASADO EN LA CORRESPONDENCIA PARCIAL (PMX)

El PMX lo introdujeron Goldberg y Lingle (1985). En él, una parte de la ristra representando a uno de los padres, se hace corresponder con una parte, de igual tamaño, de la ristra del otro padre, intercambiándose la información restante. Por ejemplo, si consideramos los dos padres siguientes:

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

el operador PMX crea las giras descendientes de la siguiente manera. En primer lugar, selecciona con probabilidad uniforme dos puntos de corte a lo largo de las ristras que representan las giras padres. Supongamos que el primer punto de corte se selecciona entre el tercer y el cuarto elemento de la gira, y el segundo entre el sexto y el séptimo elemento:

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

Se considera que existe una correspondencia biunívoca entre los elementos que forman parte de las subristras comprendidas entre los puntos de corte. En nuestro ejemplo la correspondencia establecida es la siguiente: $4 \leftrightarrow 1$, $5 \leftrightarrow 6$ y $6 \leftrightarrow 8$. A continuación la subristra del primer padre se copia en el segundo hijo. De forma análoga, la subristra del segundo padre se copia en el primer hijo, obteniéndose:

descendiente 1: (x x x | 1 6 8 | x x) y
descendiente 2: (x x x | 4 5 6 | x x).

En el siguiente paso el descendiente i-ésimo (i=1,2) se rellena copiando los elementos del i-ésimo padre. En el caso de que una ciudad esté ya presente en el descendiente, se reemplaza teniendo en cuenta la correspondencia anterior. Por ejemplo el primer elemento del descendiente 1 será un 1 al igual que el primer elemento del primer padre. Sin embargo, al existir un 1 en el descendiente 1 y teniendo en cuenta la correspondencia $1 \leftrightarrow 4$, se escoge la ciudad 4 como primer elemento del descendiente 1. El segundo, tercer y

séptimo elementos del descendiente 1 pueden escogerse del primer padre. Sin embargo, el último elemento del descendiente 1 debería ser un 8, ciudad ya presente. Teniendo en cuenta las correspondencias 8 ↔ 6, y 6 ↔ 5, se escoge en su lugar un 5. Y así:

descendiente 1: (4 2 3 | 1 6 8 | 7 5) y
descendiente 2: (3 7 8 | 4 5 6 | 2 1).

II. OPERADOR DE CRUCE BASADO EN CICLOS (CX)

El operador CX (Oliver y col., 1987) crea un descendiente a partir de los padres, de tal manera que cada posición se ocupa por el correspondiente elemento de uno de los padres. Por ejemplo, considerando los padres

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

escogemos el primer elemento del descendiente bien del primer elemento del primer padre o del primer elemento del segundo padre. Por tanto, el primer elemento del descendiente debe ser un 1 o un 2. Supongamos que escogemos el 1. Por el momento, el descendiente tendría la siguiente forma:

(1 _ _ _ _ _ _ _).

A continuación debemos de considerar el último elemento del descendiente. Ya que dicho elemento debe ser escogido de uno de los padres, tan sólo puede tratarse de un 8 o un 1. Al haber sido seleccionado el 1 con anterioridad, se escoge el 8, con lo cual el descendiente estar constituido por

(1 _ _ _ _ _ _ 8).

De forma análoga encontramos que el segundo y cuarto elemento del descendiente deben de ser seleccionados del primer padre, lo cual resulta

(1 2 _ 4 _ _ _ 8).

Una vez concluido ese ciclo, consideramos a continuación el tercer elemento del descendiente. Dicho elemento puede ser escogido de cualquiera de los padres. Supongamos que lo seleccionamos del segundo padre. Esto implica que los elementos quinto, sexto y séptimo del descendiente deben de escogerse del segundo padre, ya

que constituyen un ciclo. De ahí que se obtenga el siguiente descendiente

(1 2 6 4 7 5 3 8).

III. OPERADOR DE CRUCE BASADO EN EL ORDEN (OX 1)

El operador OX1 propuesto por Davis (1985), construye descendientes escogiendo una subgira de un padre y preservando el orden relativo de las ciudades del otro padre. Por ejemplo, considerando las dos giras padres anteriores:

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

y suponiendo que se escoge un primer punto de corte entre el segundo y el tercer elemento y un segundo punto entre el quinto y el sexto elemento, se tiene

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

Los descendientes se crean de la siguiente manera. En primer lugar, las subgiras comprendidas entre los puntos de corte se copian en los descendientes, obteniéndose

(_ _ | 3 4 5 | _ _ _) y (_ _ | 6 8 7 | _ _ _).

A continuación, comenzando por el segundo punto de corte de uno de los padres, el resto de las ciudades se copian en el orden en el que aparecen en el otro padre, omitiéndose las ciudades ya presentes. Cuando se llega al final de la ristra de la gira padre, se continúa en su primera posición. En nuestro ejemplo, esto da origen a los siguientes hijos:

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

IV. OPERADOR DE CRUCE BASADO EN EL ORDEN (OX 2)

Syswerda (1991) sugiere, en conexión con problemas de secuenciación de tareas, el operador OX2, el cual puede considerarse como una modificación del OX1. El operador OX2 selecciona al azar varias posiciones en el string de uno de los padres, para a continuación imponer en el otro padre, el orden de los elementos en las posiciones seleccionadas. Por ejemplo consideremos los padres

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

y supongamos que en el segundo padre se seleccionan las posiciones segunda, tercera y sexta. Los elementos en dichas posiciones son 4, 6 y 5 respectivamente. En el primer padre dichos elementos se encuentran en las posiciones cuarta, quinta y sexta. El descendiente coincidirá con el primer padre si exceptuamos las posiciones cuarta, quinta y sexta:

(1 2 3 _ _ _ 7 8).

A continuación rellenamos los huecos del descendiente teniendo en cuenta el orden con el que aparecen en el segundo padre. Como resultado obtenemos

(1 2 3 4 6 5 7 8).

Cambiando el papel entre el primer y segundo padre, y utilizando las mismas posiciones seleccionadas, obtendremos el segundo descendiente:

(2 4 3 8 7 5 6 1).

V. OPERADOR DE CRUCE BASADO EN LA POSICIÓN (POS)

Syswerda (1991) propone también en conexión con problemas de secuenciación, una segunda modificación del operador OX1: el operador POS. El operador POS también comienza seleccionando al azar un conjunto de posiciones en las giras padres. Sin embargo este operador impone la posición de los elementos seleccionados, en los correspondientes elementos del otro padre. Por ejemplo, si consideramos los padres

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

y suponemos que se seleccionan las posiciones segunda, tercera y sexta, esto nos proporcionaría los siguientes descendientes:

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

VI. OPERADOR DE CRUCE BASADO EN LA COMBINACIÓN DE ARCOS (ER)

Este operador desarrollado por Whitley y col. (1991), Whitley y Starkweather (1990), utiliza una "conexión de arcos", la cual proporciona para cada ciudad los arcos de los padres que comienzan o finalizan en ella. Por ejemplo, si consideramos las giras:

(1 2 3 4 5 6) y (2 4 3 1 5 6),

la "conexión de arcos" correspondiente puede consultarse en la Tabla 12.

ciudad	ciudades conectadas
1	2, 6, 3, 5
2	1, 3, 4, 6
3	2, 4, 1
4	3, 5, 2
5	4, 6, 1
6	1, 5, 2

Tabla 12: Conexión de arcos para las giras (1 2 3 4 5 6) y (2 4 3 1 5 6)

El operador ER funciona de acorde con el siguiente algoritmo:

1) Escoger la ciudad inicial de una de las dos giras padres. Esta selección puede realizarse al azar o de acorde con el criterio expuesto en el paso 4. La ciudad seleccionada se denominaría "ciudad de referencia".

2) Quitar todas las ocurrencias de la "ciudad de referencia" de la parte derecha de la tabla de "conexión de arcos" correspondiente.

3) Si la "ciudad de referencia" tiene entradas en la lista de arcos se irá al paso 4, en caso contrario al paso 5.

4) Determinar la ciudad que perteneciendo a la lista de ciudades conectadas con la "ciudad de referencia" tenga el menor número de entradas en su lista de arcos. Dicha ciudad se convierte en la nueva "ciudad de referencia". Los empates se dilucidan al azar. Ir al paso 2.

5) Si no quedan ciudades por visitar, parar el algoritmo. En otro caso, escoger al azar una ciudad no visitada e ir al paso 2.

Para el ejemplo anterior obtenemos:

- La primera gira descendiente se inicializa con una de las dos ciudades iniciales de sus giras padres. Las ciudades iniciales 1 y 2 ambas tienen 4 arcos; escogemos al azar la ciudad 2.

- La lista de ciudades para la ciudad 2 indica que los candidatos para convertirse en la siguiente "ciudad de referencia" son las ciudades 1, 3, 4 y 6. Las ciudades 3, 4 y 6 tienen todas 2 arcos: los tres iniciales menos la conexión con la ciudad 2. La

ciudad 1 tiene tres arcos y por tanto no se tiene en cuenta. Supongamos que se escoge al azar la ciudad 3.

- La ciudad 3 está conectada con la ciudad 1 y con la ciudad 4. Se escoge la ciudad 4 ya que es la que menos arcos tiene.

- La ciudad 4 tan sólo tiene un arco, a la ciudad 5, la cual se escoge a continuación como nueva "ciudad de referencia".

- La ciudad 5 tiene arcos a las ciudades 1 y 6, las cuales tienen ambas tan sólo 1 arco. Escogemos al azar la ciudad 1.

- La ciudad 1 debe ir a la ciudad 6.

La gira resultante es (2 3 4 5 1 6), la cual ha sido creada totalmente utilizando arcos tomados de las dos giras padres.

VII. OPERADOR DE CRUCE BASADO EN LA ALTERNANCIA DE LAS POSICIONES (AP)

El operador AP (Larrañaga y col., 1999), crea un descendiente seleccionando ciudades alternativamente del primer y segundo padre en el orden ocupado por los mismos, omitiéndose aquellas ciudades que ya se encuentran presenten en la gira descendiente. Por ejemplo, si los padres son

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

el operador AP proporciona el siguiente descendiente (1 3 2 7 5 4 6 8) y cambiando el orden de los padres se obtiene (3 1 7 2 5 4 6 8).

3.2.1.B) Operadores de mutación

A. OPERADOR DE MUTACIÓN BASADO EN EL DESPLAZAMIENTO (DM)

El operador DM (Michalewizc, 1992) comienza seleccionando una subristra al azar. Dicha subristra se extrae de la gira, y se inserta en un lugar aleatorio. Por ejemplo, si consideramos la gira representada por

(1 2 3 4 5 6 7 8),

y suponemos que se selecciona la subristra (3 4 5), después de quitar dicha subristra tenemos (1 2 6 7 8). Supongamos que aleatoriamente

seleccionamos la ciudad 7 para insertar a partir de ella la subgira extraida. Esto produciría la gira:

(1 2 6 7 3 4 5 8).

B. OPERADOR DE MUTACIÓN BASADO EN CAMBIOS (EM)

El operador EM (Banzhaf, 1990) selecciona al azar dos ciudades en la gira y las cambia. Por ejemplo, si consideremos la gira representada por

(1 2 3 4 5 6 7 8),

y suponemos que seleccionamos al azar la tercera y la quinta ciudad. El resultado del operador EM sobre la gira anterior sería

(1 2 5 4 3 6 7 8).

C. OPERADOR DE MUTACIÓN BASADO EN LA INSERCIÓN (ISM)

El operador ISM (Fogel, 1993; Michalewizc, 1992) escoge aleatoriamente una ciudad en la gira, para a continuación extraer dicha ciudad de la gira, e insertarla en un lugar seleccionado al azar. Por ejemplo, si consideramos de nuevo la gira

(1 2 3 4 5 6 7 8),

y suponiendo que se seleccione la ciudad 4, para colocarla a continuación de la ciudad 7, el resultado sería

(1 2 3 5 6 7 4 8).

D. OPERADOR DE MUTACIÓN BASADO EN LA INVERSIÓN SIMPLE (SIM)

El operador SIM (Holland, 1975; Grefenstette y col., 1985) selecciona aleatoriamente dos puntos de corte en la ristra, para a continuación invertir la subristra comprendida entre ambos. Por ejemplo, si consideramos la gira

(1 2 3 4 5 6 7 8),

y suponemos que el primer punto de corte se escoge entre la segunda y tercera ciudad, y el segundo punto de corte se escoge entre la quinta y la sexta ciudad, la gira resultante sería

(1 2 5 4 3 6 7 8).

E. OPERADOR DE MUTACIÓN BASADO EN LA INVERSIÓN (IVM)

El operador IVM (Fogel, 1988, 1993) es similar al operador DM. Se selecciona al azar una subgira, para a continuación y una vez extraída la misma, insertarla en orden contrario en una posición seleccionada aleatoriamente. Por ejemplo, si consideramos la gira

(1 2 3 4 5 6 7 8),

y se supone que se escoge la subgira (3 4 5), para insertarla a continuación de la ciudad 7, obtendríamos

(1 2 6 7 5 4 3 8).

F. OPERADOR DE MUTACIÓN BASADO EN EL CAMBIO (SM)

Este operador de mutación, introducido por Syswerda (1991), selecciona una subgira al azar y a continuación cambia el orden de las ciudades de la misma. Por ejemplo, considerando la gira

(1 2 3 4 5 6 7 8),

y suponiendo que se escoge la subgira (4 5 6 7) podríamos obtener como resultado

(1 2 3 5 6 7 4 8).

3.2.2. REPRESENTACIÓN BINARIA

En una representación binaria de un TSP con n ciudades, cada ciudad se codifica como una subristra de $[\log_2 n]$ bits, donde $[\log_2 n]$ denota la suma entre la parte entera del logaritmo en base 2 de n y la unidad. Una gira de n ciudades, se representará por medio de una ristra de $[\log_2 n]$ bits. Por ejemplo, en un TSP de 6 ciudades, las ciudades se representan por medio de subristras de 3 bits (véase la Tabla 13).

i	ciudad i	i	ciudad i
1	000	4	011
2	001	5	100
3	010	6	101

Tabla 13: Representación binaria para un TSP con 6 ciudades

Siguiendo la representación binaria definida en la tabla la gira 1−2−3−4−5−6 se representa por medio de

(000 001 010 011 100 101).

Nótese que existen subristras de 3 bits que no corresponden a ninguna ciudad, como por ejemplo 110 y 111.

3.2.2.A) El cruce clásico

Para mostrar la problemática que surge al aplicar el operador de cruce basado en un punto, se consideran las dos giras siguientes:

(000 001 010 011 100 101) y (101 100 011 010 001 000).

Suponiendo que el punto de cruce escogido al azar se encuentre entre el noveno y el décimo bit, tendríamos:

(000 001 010 | 011 100 101) y (101 100 011 | 010 001 000).

La combinación de las distintas partes da como resultado

(000 001 010 010 001 000) y (101 100 011 011 100 101).

Ninguna de las ristras anteriores representan giras legales.

3.2.2.B) La mutación clásica

Al aplicar el operador de mutación, cada bit tiene una probabilidad de ser alterado
cercana a cero. Por ejemplo, si consideramos de nuevo la ristra

(000 001 010 011 100 101),

y suponemos que se mutan el primer y segundo bit, obtenemos como resultado la ristra:

(110 001 010 011 100 101),

la cual no representa una gira.

3.2.2.C) Otro tipo de representación binaria

Whitley y col. (1989, 1991), sugieren una representación binaria diferente de la anterior, en la cual en primer lugar se define una lista ordenada que contiene todos los posibles arcos no dirigidos. Con la ayuda de esta lista, una gira puede escribirse como un vector binario, con una longitud igual al número de arcos en

la lista anterior. El i-ésimo elemento del vector binario valdrá 1, si y sólo si, el i-ésimo arco de la lista ordenada es parte de la gira. En caso contrario su valor será 0. Por ejemplo, para el TSP de 6 ciudades, la lista ordenada podría definirse a partir de la siguiente lista de arcos no dirigidos:

(1, 2), (1, 3), (1, 4), (1, 5), (1, 6), (2, 3), (2, 4), (2, 5), (2, 6), (3, 4), (3, 5), (3, 6), (4, 5), (4, 6), (5, 6).

Entonces la gira 1 − 2 − 3 − 4 − 5 − 6, se representa por el vector binario

100011000100101.

3.2.3. UTILIZACIÓN DE ALGORITMOS GENÉTICOS PARALELOS EN EL TSP

Es bastante interesante la implementación de un algoritmo genético paralelo utilizado en la resolución de problemas de TSP para poblaciones de gran tamaño. Dicha implementación está descrita en Muhlenbein et al. 87, Mulhenbein et al. 88, Gorges-Schleuter 89, Mulhenbein 91.

En este caso cada procesador es responsable de un único cromosoma. Como los procesadores (cromosomas) están unidos entre ellos mediante una topología fijada, la población de cromosomas está estructurada. A continuación se establece un entorno alrededor de cada cromosoma basándose en dicha topología, y la reproducción tiene lugar entre los cromosomas de dicho entorno únicamente. Como los entornos no tienen por qué ser disjuntos, el nuevo material genético se va difundiendo lentamente a través de la población entera. El beneficio principal de esta forma de organización en una población uniforme, donde cada cromosoma puede 'aparearse' con cualquier otro, es que la diversidad se mantiene más fácilmente en la población. Además, el algoritmo genético paralelo puede utilizarse también con poblaciones pequeñas, sin el problema de la convergencia prematura.

El algoritmo genético paralelo puede describirse como sigue:

- Se crea una población inicial aleatoria.
- Se selecciona un entorno alrededor de cada cromosoma.
- Cada cromosoma selecciona un compañero para 'aparearse' en su entorno.
- Se crea un descendiente utilizando un operador de cruzamiento adecuado (normalmente se suele utilizar un operador de cruce basado en el orden).
- Se evalúa la función de adaptación (función fitness) del descendiente y dicho descendiente reemplazará a sus progenitores si se cumplen las dos condiciones siguientes:

a) Es mejor que el peor cromosoma del entorno de sus padres.
b) Es mejor que el padre, si el padre es el mejor cromosoma de su propio entorno.

Este algoritmo es totalmente asíncrono. Cada cromosoma selecciona un compañero para unirse a él en su entorno sin considerar la población como un conjunto (lo que significa que algunos cromosomas pueden estar en la generación T, mientras que otros están en la población T+1 o superior. Notar también que el algoritmo siempre busca un nuevo óptimo local. En particular, combina dos óptimos locales para formar un nuevo óptimo local que va a ser mejor que los dos previos.

Se ha comprobado que con este tipo de algoritmos se obtienen muy buenos resultados a la hora de resolver el problema del agente viajero (e incluso otros muchos problemas)

3.3. UNA APLICACIÓN DE LOS ALGORITMOS GENÉTICOS AL PROBLEMA DEL CLUSTERING JERÁRQUICO

En esta aplicación se plantea el uso de los Ags como herramienta para resolver el problema del clustering jerárquico. Partiendo de la matriz de disimilaridad de los datos, tratamos de hallar, usando los Ags, la disimilaridad ultramétrica más cercana a la disimilaridad de los datos, lo cual nos conduce a la mejor clasificación jerárquica para los mismos. Plantearemos dos aproximaciones diferentes: una basada en la utilización del orden de los datos y otra basada en la penalización de la función objetivo.

En primer lugar vamos a introducir los conceptos más importantes referidos al clustering jerárquico y en qué consiste dicho problema.

3.3.1. EL CLUSTERING JERÁRQUICO

El problema del clustering consiste en, dado un conjunto de datos, descubrir la estructura de grupos que se encuentra en dicho conjunto, si es que existe. Las técnicas de clustering más utilizadas son dos: el clustering particional y el clustering jerárquico.

El clustering particional consiste en dividir el conjunto de datos en grupos, de manera que los datos que se encuentren en un grupo sean lo más parecidos entre sí, a la vez que lo más diferentes posible a los datos que se encuentren en los demás grupos. Es decir, se trata de dar una partición del conjunto de datos.

El clustering jerárquico es otra técnica de clustering que en vez de crear una única partición, crea una sucesión encajada de particiones cuya estructura puede ser representada por medio de un árbol. Un ejemplo de cluster jerárquico puede verse en la Ilustración 9.

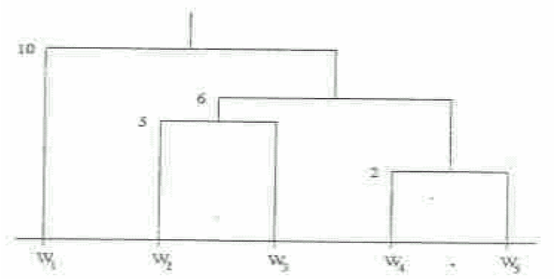

Ilustración 9: Un ejemplo de una clasificación jerárquica

Existen principalmente dos tipos de algoritmos para llevar a cabo el clustering jerárquico: los algoritmos aglomerativos y los algoritmos divisivos. Los primeros, que son los más utilizados, parten de una situación en la que cada dato del conjunto se encuentra en un cluster y en cada paso del algoritmo se juntan los dos clusters "más cercanos", hasta tener un único cluster donde están todos los datos. Las diferentes formas de elegir esos clusters "más cercanos" generan diferentes algoritmos. Por otro lado, los algoritmos divisivos realizan la tarea contraria, es decir, parten de una situación en la que existe un único cluster con todos los datos, y en cada paso del algoritmo se divide uno de los clusters en dos. El algoritmo termina cuando cada cluster está compuesto de un único elemento del conjunto.

Ambos algoritmos tienen una característica en común: la forma de construir el árbol es local. Esto nos hace pensar en intentar mejorar la clasificación que se obtiene con ellos mediante un algoritmo de búsqueda global como son los algoritmos genéticos. Supondremos que la información de entrada al algoritmo va a ser una matriz cuadrada simétrica que representa las distancias entre los datos y que llamaremos matriz de disimilaridad.

Ahora definiremos formalmente lo que es una clasificación jerárquica así como otros elementos que intervendrán en nuestro problema.

- Un árbol en un conjunto Ω es un subconjunto T de P (Ω) tal que se cumple:
  1) $\Omega \in T$;
  2) $\phi \notin T$;
  3) $\forall w \in \Omega \Rightarrow \{ w \} \in T$;
  4) $\forall A, B \in T \Rightarrow A \cap B \in \{ \phi , A , B \}$

- Formalmente una clasificación jerárquica o dendograma es un par (T, h) formado por árbol T junto con una aplicación h definida en los nodos de T de manera que se cumple:
  1) h (A) = 0 $\Leftrightarrow$ A = { w } para algún w $\in \Omega$
  2) si A $\cap$ B $\neq \phi$ entonces A $\subset$ B $\Leftrightarrow$ h (A) $\leq$ h (B)

- A partir de un dendograma es posible construir una nueva disimilaridad en los datos de la siguiente manera: U (w, w') = $\min_{A \in T}$ { h (A) / w, w' $\in$ A}.

Esta disimilaridad posee unas propiedades especiales:

i. $\forall$ w, w' $\in \Omega \Rightarrow U$ (w, w') = U (w', w)
ii. $\forall$ w $\in \Omega \Rightarrow U$ (w, w) = 0
iii. $\forall$ w, w', w" $\in \Omega \Rightarrow U$ (w, w") $\leq$ max {U (w, w'), U (w', w")}

La última propiedad se denomina propiedad ultramétrica y las disimilaridades que cumplen estas tres condiciones, disimilaridades ultramétricas. Se puede probar que existe una biyección entre disimilaridades ultramétricas y clasificaciones jerárquicas. Este hecho nos permite medir la bondad de una clasificación jerárquica (basta utilizar alguna medida que calcule la diferencia entre la disimilaridad inicial de los datos y la disimilaridad ultramétrica de la clasificación obtenida) y nos permite realizar la búsqueda en el conjunto de las disimilaridades ultramétricas en lugar de hacerlo directamente en el de las clasificaciones jerárquicas.

- La norma que utilizaremos para calcular la diferencia entre las disimilaridades será la norma Euclídea L_2.

Por lo tanto nuestro problema lo podemos escribir del siguiente modo:
"Dado un conjunto de datos con una matriz de disimilaridad D, hallar la disimilaridad ultramétrica U más cercana a D en norma L_2".

3.3.2. APROXIMACIONES AL PROBLEMA USANDO ALGORITMOS GENÉTICOS.

Para resolver el problema se utilizarán dos aproximaciones distintas: la primera la denominaremos *aproximación basada en el orden* y la siguiente *aproximación basada en la penalización*.

a) Aproximación basada en el orden.

Aquí nos basaremos en dos resultados. El primero fue demostrado por Lerman y dice lo siguiente:

" Si Ω es un conjunto de datos y $U = [\ \delta_{i,j}\ ]_{i,j=1,\ldots,n}$ es una disimilaridad ultramétrica sobre los mismos entonces existe un orden Θ para los datos, tal que

si ordenamos las filas y columnas de U en este orden, la matriz cumple las siguientes propiedades:

- $\forall\, i \le j \Rightarrow \delta_{i,j} \le \delta_{i,j+1}$

- $\forall\, k \in \{1, 2, \ldots, n\}$ si $\delta_{k,k+1} = \delta_{k,k+2} = \ldots = \delta_{k,k+s+1} \le \delta_{k,k+s+2}$ entonces:
  - $\delta_{k+1,j} \le \delta_{k,j} \qquad \forall\, k+1 < j \le k+s+1$
  - $\delta_{k+1,j} = \delta_{k,j} \qquad \forall\, j > k+s+1$ ”

El otro resultado, demostrado por Chandon, apunta lo siguiente:

“ Si denotamos por $S = \{ (i, j) / w_i \neq w_j \}$ el conjunto de pares de elementos diferentes de Ω, entonces cualquier disimilaridad ultramétrica $U = [\delta_{i,j}]_{i,j=1,\ldots,n}$ puede ser descompuesta en una relación de preorden Γ definida en S $(\Gamma_1 < \Gamma_2 < \ldots < \Gamma_k)$ y una sucesión de números $\delta_1 < \delta_2 < \ldots < \delta_k$ tal que $\forall\, (i, j) \in \Gamma_r \Rightarrow \delta_{i,j} = \delta_r$ ”.

Se demuestra también que si la disimilaridad ultramétrica es la disimilaridad solución de nuestro problema, entonces se cumple que:

$$\delta_r = (1 / n_r) \sum_{(i,j)\in\Gamma r} d_{i,j} \qquad \forall\, r \in \{1, 2, \ldots, k\}$$

donde $d_{i,j}$ está denotando la disimilaridad inicial entre los datos w_i y w_j, y n_r es el número de pares de Γ_r .

Dado un orden para el conjunto de datos Θ y una estructura de árbol T es muy fácil hallar la disimilaridad ultramétrica más cercana a los datos con este orden y esta estructura de árbol. La estructura de árbol contiene la información de los elementos iguales de la matriz de disimilaridad buscada. El problema por lo tanto puede ser planteado como un problema de programación cuadrática en n-1 variables con restricciones.

Esto se puede ver de forma sencilla en el siguiente ejemplo. Supongamos que el conjunto de datos Ω tiene 5 elementos y que la matriz de disimilaridad inicial para los datos en el orden Θ es:

$$\begin{pmatrix} 0 & 4 & 3 & 5 & 1 \\ & 0 & 2 & 8 & 3 \\ & & 0 & 6 & 2 \\ & & & 0 & 9 \\ & & & & 0 \end{pmatrix}$$

siendo la estructura de árbol la dada en la Figura 1 que vimos antes. La matriz ultramétrica óptima tiene el siguiente aspecto

$$\begin{pmatrix} 0 & \delta_1 & \delta_1 & \delta_1 & \delta_1 \\ & 0 & \delta_2 & \delta_3 & \delta_3 \\ & & 0 & \delta_3 & \delta_3 \\ & & & 0 & \delta_4 \\ & & & & 0 \end{pmatrix}.$$

El problema cuadrático se plantea como:

$$\min\ (\delta_1 - \frac{(4+3+5+1)}{4})^2 + (\delta_2 - 2)^2 + \ (\delta_3 - \frac{(8+3+6+2)}{4})^2 + (\delta_4 - 9)^2$$

sujeto a las restricciones siguientes:

$\delta_3 - \delta_1 \leq 0$; $\delta_2 - \delta_3 \leq 0$; $\delta_4 - \delta_3 \leq 0$

$\delta_i \in \Re^+$ y $\delta_i \geq 0$ $i = 1, 2, 3, 4$

Por lo tanto, un individuo de nuestro algoritmo genético va a codificar al mismo tiempo un orden para los datos y una estructura de árbol. Nuestros individuos van a ser permutaciones de los números del 1 al $n + (n - 2)$, donde los enteros del 1 al n están representando el orden de los elementos del conjunto y los números del $n + 1$ hasta el $2n - 2$ están codificando la estructura del árbol. En nuestro algoritmo, a la hora de evaluar un individuo, en vez de resolver el problema de programación cuadrática en cada paso y calcular su distancia con la disimilaridad inicial de los datos, la disimilaridad ultramétrica para el orden y la estructura de árbol se calcula de forma aproximada. De esta forma ganamos bastante tiempo computacional.

Cada variable de disimilaridad ultramétrica es calculada como la media de la disimilaridad inicial de los datos correspondientes a esa variable. Así, en nuestro ejemplo estos valores son:

$$\delta_1 = \frac{(4+3+5+1)}{4} = 3.25 \qquad \delta_2 = 2$$
$$\delta_3 = \frac{(8+3+6+2)}{4} = 4.75 \qquad \delta_4 = 9.$$

Sin embargo, como puede comprobarse, de esta forma no siempre se obtiene una disimilaridad ultramétrica. Para conseguirla, fijamos el valor de la variable que se supone debería ser mayor, y se fuerza a las demás variables en orden a que cumplan las restricciones del programa cuadrático. Así, obtendríamos finalmente:

$$\delta_1 = \frac{(4+3+5+1)}{4} = 3.25 \qquad \delta_2 = 2$$
$$\delta_3 = \frac{(8+3+6+2)}{4} = 3.25 \qquad \delta_4 = 3.25.$$

Esta forma de evaluar al individuo tiene como ventaja que el esfuerzo computacional es mucho menor que si tuviéramos que resolver el programa cuadrático en cada paso.

b) Aproximación basada en la penalización.

La aproximación basada en la penalización se basa en buscar disimilaridades cercanas a la disimilaridad de los datos y en el caso de que éstas no sean ultramétricas penalizar a la función objetivo. De esta manera, cada individuo codificará de alguna forma una disimilaridad U , y la función objetivo vendrá dada por:

$$\Phi(U) = L(U) + P(U)$$

donde $L(U) = \sum_{i<j} (d_{i,j} - \delta_{i,j})^2$ mide la distancia entre la disimilaridad inicial

de los datos y la disimilaridad propuesta

y $P(U) = \sum_{\Lambda} (\delta_{i,k} - \delta_{j,k})^2$ con $\Lambda = \{ (i, j, k) : \delta_{i,j} \leq \min (\delta_{i,k}, \delta_{j,k})\}$

y mide la penalización (en cuánto está violando la disimilaridad propuesta *U* las condiciones de ultramétrica).

Dado que sabemos que en una disimilaridad ultramétrica únicamente puede haber n – 1 números diferentes, vamos a llevar a cabo un clustering particional en los elementos de la matriz de disimilaridad inicial de los datos en un número de clusters $t \leq n - 1$. La disimilaridad propuesta será entonces aquella que se forma al sustituir cada elemento de la matriz de disimilaridad inicial por el valor medio de los elementos que pertenecen al cluster en el que se encuentra dicho elemento. De esta forma cada individuo del AG va a estar compuesto por una permutación de los números del 1 al (n(n+1)/2)+ n – 2 , donde los números del 1 al n(n+1)/2 representan los elementos de la matriz de disimilaridad inicial y los números de n(n+1)/2+1 al (n(n+1)/2)+ n – 2 actúan de separadores para determinar los clusters.

Suponiendo como matriz de disimilaridad inicial de los datos la dada en el ejemplo anterior, el individuo siguiente: (2 4 5 11 10 12 8 1 9 13 3 6) se codificaría de la siguiente manera. Primero los clusters que se obtienen son cuatro, esto es: {2, 4, 5} {10} {8, 1, 9} {3, 6}, siendo el valor medio de los elementos de cada cluster: $\delta_1 = 2$; $\delta_2 = 9$; $\delta_3 = 4$; $\delta_4 = 6{,}5$ y la matriz de disimilaridad que se obtiene es:

$$\begin{pmatrix} 0 & 4 & 2 & 6.5 & 2 \\ & 0 & 2 & 6.5 & 3 \\ & & 0 & 4 & 4 \\ & & & 0 & 9 \\ & & & & 0 \end{pmatrix}.$$

3.3.3. EL ALGORITMO GENÉTICO UTILIZADO

El AG utilizado para realizar el experimento es un “steady-state”. Este algoritmo se diferencia principalmente del algoritmo genético canónico en que en vez de evolucionar población a población, evoluciona individuo a individuo. En cada paso del algoritmo se eligen dos individuos, estos son cruzados y se toma uno de los resultantes. Dicho individuo es evaluado. Si su valor de función objetivo es mejor que el correspondiente al peor individuo de la población, entonces se introduce en ella, de otro modo se deshecha. Otra característica de este tipo de AG es que converge hacia el óptimo.

Dado que los individuos de los AGs diseñados son permutaciones de números, los operadores que se utilizarán son los mismos que se desarrollaron para aplicarlos en el problema del agente viajero (en concreto, como operadores de cruce se utilizaron: CX, PMX, AP, OX1, y como operadores de mutación: SM, ISM, SIM, IVM, EM y DM).

Para comparar los algoritmos se llevaron a cabo varios experimentos. Los conjuntos utilizados son dos: el primero es un conjunto de tamaño 18 representando flores de jardín; el segundo es el conjunto de datos Ruspini, clásico en la comparación de métodos clustering y que consta de 75 puntos bidimensionales.

Los parámetros utilizados por el algoritmo fueron los siguientes:

- A la probabilidad de cruce le fue dado un valor de 1.
- El tamaño de la población fue establecido en función del tamaño *l* del individuo de manera que se le asignó un valor de 3**l*.
- La probabilidad de mutación también fue establecida en función de *l* con un valor de 1 / *l*.
- Se utilizaron dos criterios de terminación: el primero consistía en parar el algoritmo si el mejor individuo de la población no se modificaba en 50.000 generaciones; el segundo consistió en no dejar sobrepasar el algoritmo cierto número de iteraciones (para la aproximación basada en el orden este número fue 150.000 en el conjunto de flores y 300.000 en el Ruspini; para la aproximación basada en la penalización, en el conjunto de flores dicho número se estableció en 100.000).

En el primer conjunto se usaron ambas aproximaciones, y para medir la efectividad de los diferentes operadores se llevaron a cabo 10 experimentos por cada aproximación con cada combinación de operadores de cruce y mutación, es decir, para cada aproximación y cada operador de cruce se realizaron 60 experimentos, y para cada aproximación y cada operador de mutación se realizaron 40.

En las Tablas 14 o 15 se pueden ver los resultados obtenidos para cada aproximación y cada operador. En dichas tablas se representa la diferencia cuadrática entre la disimilaridad propuesta por el algoritmo y la disimilaridad inicial de los datos.

	Basada en el Orden	
	Media	Mínimo
PMX	2.266.480	1.948.069
CX	1.978.757	1.942.537
OX1	2.136.466	1.948.155
AP	2.403.217	1.970.875
SM	2.236.766	1.942.917
SIM	2.232.747	1.948.240
ISM	2.272.982	1.945.578
IVM	2.151.611	1.942.537
EM	2.115.374	1.942.537
DM	2.167.900	1.945.492

Tabla 14: Resultados de la aproximación basada en el orden con el primer conjunto de datos

	Basada en la Penalización	
	Media	Mínimo
PMX	3.290.859	2.887.105
CX	3.215.737	3.020.156
OX1	3.191.071	2.956.487
AP	3.394.806	2.905.927
SM	3.368.590	2.956.487
SIM	3.229.441	2.924.306
ISM	3.365.626	3.128.738
IVM	3.229.217	2.996.016
EM	3.220.158	3.029.918
DM	3.225.678	2.887.105

Tabla 15: Resultados de la aproximación basada en la penalización con el primer conjunto de datos

Observamos que con la aproximación basada en el orden se obtienen mucho mejores resultados que con la basada en la penalización. Para comparar estos resultados se utilizaron los algoritmos de clustering jerárquicos implementados en el programa comercial SAS que producían, tras su aplicación, una matriz de disimilaridad ultramétrica. La Tabla 16 recoge las diferencias cuadráticas entre las matrices de disimilaridad ultramétrica obtenidas por estos algoritmos y la inicial:

Métodos	Single	Average	Complete
Resultados	51.644.604	49.918.336	51.972.852
Métodos	Flexible	Mc.Quitty	Ward
Resultados	64.569.380	42.249.532	50.365.488

Tabla 16: Resultados obtenidos por los métodos SAS en el primer conjunto

Observamos que claramente cualquiera de nuestras aproximaciones supera con creces los resultados obtenidos por los algoritmos clásicos.

Para el segundo conjunto se realizó únicamente la aproximación basada en el orden ya que la otra exigía una cantidad bastante grande de recursos computacionales. Los resultados con dicho algoritmo y con el programa SAS fueron los siguientes:

	Basada en el Orden	
	Media	Mínimo
PMX	36.422.418	26.570.410
CX	34.793.051	28.710.376
OX1	34.378.658	27.794.166
AP	54.254.448	41.654.588
SM	41.204.050	29.015.210
SIM	39.485.054	27.256.516
ISM	43.268.791	29.295.194
IVM	38.795.052	28.710.376
EM	37.807.301	27.437.560
DM	38.826.927	26.570.410

Tabla 17: Resultados obtenidos en el conjunto Ruspini

Métodos	Single	Average	Complete
Results	486.150.880	351.939.904	413.378.016
Methods	Flexible	Mc.Quitty	Ward
Results	518.739.814	351.883.584	465.034.027

Tabla 18: Resultados obtenidos por el método de SAS en el conjunto Ruspini

De nuevo se observa que el algoritmo propuesto obtiene mucho mejores resultados que los algoritmos clásicos, no sólo en conjuntos pequeños sino en conjuntos de tamaño grande.

3.4. UNA APLICACIÓN DE LOS ALGORITMOS GENÉTICOS AL PROBLEMA DE PLANIFICACIÓN DE ACTIVIDADES

Los problemas de planificación se encuentran en una gran variedad de campos, incluyendo la manufactura y el servicio industrial. Los problemas de planificación son numerosos y variados. En términos más amplios, la planificación implica el reparto de recursos durante un período de tiempo para llevar a cabo un conjunto de actividades. En general, no hay un algoritmo general que garantice dar una solución óptima y hacerlo en un tiempo polinomial. Por tanto, los problemas de planificación son principales candidatos para la aplicación de la tecnología de la Inteligencia Artificial.

Consideremos un entorno de fábrica en el cual n actividades han de ser procesadas por m máquinas. Cada actividad tendrá un conjunto de restricciones sobre el uso de las máquinas y un tiempo de procesamiento para cada máquina. Nuestro problema consiste en encontrar la secuencia de actividades a realizar por cada máquina con el fin de minimizar una función objetivo dada.

Fue Davis quien comenzó a investigar la utilización de AGs para aplicarlos al problema de planificación de actividades y Whitley analizó el uso de los operadores de cruzamiento. Un aspecto crucial de cada trabajo es el método utilizado para codificar las planificaciones. Recientemente se han obtenido mejores resultados rechazando los conocimientos convencionales en el uso de representaciones binarias, y empleando en su lugar una codificación directa. La aplicación de los algoritmos genéticos a la planificación considera secuencias o listas de actividades como individuos o miembros de una población. En un entorno multi-máquina, un cromosoma consiste en subcromosomas, cada uno de los cuales contiene información sobre la secuencia de trabajo de una máquina. Una mutación en un cromosoma padre es equivalente a un intercambio de un par adyacente en la correspondiente secuencia. Una lista de genotipos será simplemente una lista en la que se especificará el orden y la duración de las actividades que deben ser realizadas por cada máquina. Pero un simple cruzamiento aplicado a las cadenas puede en muchas ocasiones dar lugar a un descendiente no válido con individuos representados varias veces y el orden de realización de las actividades alterado. Debido a esto, se necesitan métodos de cruzamiento más sofisticados.

3.4.1. ALGUNOS EJEMPLOS DE PLANIFICACIÓN

Estudiaremos el mecanismo y examinaremos la eficacia del uso de AGs en una secuencia de actividades de un sistema manufacturero. Estudiaremos dos ejemplos: en el primero analizaremos el tiempo de procesamiento más corto (SPT óptimo) que minimiza el tiempo total en el caso de una única máquina; en el segundo demostraremos que el algoritmo genético descubre la regla óptima de Johnson que minimiza el tiempo total utilizado en el caso de dos máquinas.

El tamaño de la población fijado será 50. Utilizaremos el operador de cruzamiento basado en la correspondencia parcial (PMX). Las probabilidades de cruzamiento y mutación están fijadas y son 0,6 y 0,001 respectivamente. La población genética se crea usando la estrategia de selección pura, en la que cada secuencia tiene copias en la población total en proporción a su rendimiento. En cada iteración se inserta una secuencia 'hijo' en la población y se evalúan los individuos mediante la función de adaptación (función fitness). Después se elimina el peor miembro de la población. Cada nueva generación del algoritmo genético (se realizarán un máximo de 2000 generaciones) lleva consigo la creación y evaluación de 50 secuencias 'hijo'.

- EJEMPLO 1:

  En el caso de una única máquina y dado un conjunto de actividades, si lo que queremos es minimizar el tiempo total empleado, entonces la secuencia óptima viene dada por la regla secuencial SPT (tiempo de procesamiento más corto).

  Planteamiento del problema:

  Tenemos una máquina capaz de realizar una operación, y diez trabajos que requieren una única operación cada uno con procesos de tiempo variables. El objetivo de nuestra planificación es minimizar el tiempo total.

  Si hacemos una completa enumeración de todas las posibles secuencias de actividades, el número total de secuencias que podríamos evaluar sería 10!, es decir, 3628800. Alternativamente, se ha demostrado que usando algoritmos genéticos se obtiene la secuencia óptima (y por tanto la regla secuencial mencionada anteriormente) examinando un número bastante más pequeño de secuencias.

  A continuación tenemos los tiempos de proceso de las diez actividades:

Actividad	J1	J2	J3	J4	J5	J6	J7	J8	J9	J10
Tiempo de proceso	18	5	21	17	14	2	8	12	24	7

Usando la regla secuencial óptima SPT obtenemos la siguiente secuencia de actividades:

```
J6, J2, J10, J7, J8, J5, J4, J1, J3, J9
```

Los resultados de las ejecuciones del algoritmo genético vienen dados a continuación:

Número de réplicas	3	9	20	25	28	30
% de secuencias examinadas	0.2	0.5	1.0	1.5	2.0	2.5

La primera fila nos indica el número de réplicas en las cuales se obtiene la secuencia óptima; la segunda muestra el porcentaje del número total de secuencias generadas hasta obtener la secuencia óptima respecto el máximo número de secuencias posibles. Por ejemplo, en exactamente tres réplicas, el algoritmo genético es capaz de obtener la secuencia óptima examinando menos del 0.2% del número total de posibles secuencias. Notar también que incluso en el peor de los casos, el algoritmo genético requiere examinar únicamente el 2.5% de las secuencias antes de encontrar la secuencia óptima.

- EJEMPLO 2:

Dado un conjunto de actividades que necesitan ser procesadas en dos máquinas, el algoritmo de Johnson es capaz de obtener una planificación mínima que minimice el tiempo total del proceso.

Planteamiento del problema:

Tenemos dos máquinas cada una de las cuales es capaz de realizar una operación, y diez trabajos que requieren dos operaciones en las dos máquinas con procesos de tiempo diferentes. El objetivo de la planificación resultante es minimizar el tiempo total empleado en el proceso.

A continuación tenemos los tiempos de proceso de las diez actividades en ambas máquinas:

Actividades	J1	J2	J3	J4	J5	J6	J7	J8	J9	J10
Tiempo máquina 1	20	13	25	4	11	1	22	15	9	18
Tiempo máquina 2	3	7	10	21	16	5	19	8	14	23

Aplicando la regla de Johnson, la planificación óptima resultante es:

```
J6,J4,J9,J5,J10,J7,J3,J8,J2,J1.
```

El tiempo total máximo empleado en el proceso es 141.

De nuevo, si no aplicáramos el algoritmo de Johnson tendríamos que evaluar 362880 (10!) planificaciones antes de determinar la óptima. Alternativamente, se ha demostrado que usando algoritmos genéticos se puede encontrar la secuencia óptima y de este modo realizaríamos un número de evaluaciones notablemente menor. Los resultados de este experimento son los siguientes:

Número de réplicas	3	7	18	27	30
% de secuencias examinadas	0.2	0.5	1.0	1.5	2.0

La primera fila indica el número de réplicas en las que la secuencia óptima fue obtenida; la segunda muestra el porcentaje del número total de secuencias generadas para obtener la secuencia óptima. Por ejemplo, utilizando exactamente tres réplicas, el algoritmo genético fue capaz de obtener la secuencia óptima examinando menos del 0.2% del número máximo posible de secuencias. También notar que en el peor de los casos, el AG ha requerido examinar únicamente un máximo del 2% de las secuencias antes de encontrar la óptima.

3.5. CONCLUSIÓN

Cuando describimos una aplicación de los algoritmos genéticos, necesitamos considerar más aspectos que los teóricos aquí descritos. Cada aplicación necesitará su propia función de aptitud (función fitness). Muchos de los pasos de un algoritmo genético tradicional pueden implementarse utilizando un número distinto de algoritmos. Por ejemplo, la población inicial debe ser generada aleatoriamente o usando algún método heurístico.

La implementación de un algoritmo genético es bastante costosa en muchos casos, ya que posiblemente las poblaciones de soluciones se consiguen a partir de evaluaciones computacionales intensivas de la función fitness. Como ya hemos visto, una posible solución a este problema es la paralelización del proceso. Mientras esta postura no constituye ningún problema en principio, sí es cierto que requiere ciertas modificaciones en la existencia de algoritmos o la introducción de algunos nuevos con el fin de conocer las restricciones de una máquina paralela dada.

Se dice que la evolución cultural humana ha reemplazado a la biológica - que nosotros, como especie, hemos llegado a un punto en el que somos capaces de controlar conscientemente nuestra sociedad, nuestro entorno y hasta nuestros genes al nivel suficiente para hacer que el proceso evolutivo sea irrelevante. Se dice que los caprichos culturales de nuestra cambiante sociedad son los que determinan la aptitud hoy en día, en lugar de la marcha enormemente lenta, en comparación, de la mutación genética y la selección natural. En cierto sentido, esto puede ser perfectamente cierto.

Pero en otro sentido, nada podría estar más lejos de la verdad. La evolución es un proceso de resolución de problemas cuyo poder sólo comenzamos a comprender y explotar; a pesar de esto, ya está funcionando por todas partes, moldeando nuestra tecnología y mejorando nuestras vidas, y, en el futuro, estos usos no harán sino multiplicarse. Sin un conocimiento detallado del proceso evolutivo no habrían sido posibles ninguno de los incontables avances que le debemos a los algoritmos genéticos. Aquí hay una lección que deben aprender los que niegan el poder de la evolución y los que niegan que el conocimiento de ella tenga beneficios prácticos. Por increíble que pueda parecer, la evolución funciona. Como lo expresa el poeta Lord Byron: ``Es extraño pero cierto; porque la verdad siempre es extraña, más extraña que la ficción."

4. REFERENCIAS

- Banzhaf, W. (1990). "The "molecular" traveling salesman". *Biological Cybernetics*, 64, 7-14.

- Brady, R. M. (1985). "Optimization strategies gleaned from biological evolution". *Nature*, 317, 804-806.

- Chandon J. L., Lemaire, J. y Pouget, J. (1980). "Construction l'ultrametrique la plus proche d'une dissimilarité au sens des moindres carrés". *R. A. I. R. O. Recherche Operationalle,* 14, 157-170.

- Cooper, G., et al. (1997). "An evaluation of machine-learning methods for predicting pneumonia mortality". *Artificial Intelligence in Medicine.*

- Davis, L. (1985). "Applying adaptive algorithms to epistatic domains", *Proceedings of the International Joint Conference on Artificial Intelligence*, 162-164

- Fayyad, U. M., Uthurusamy, R. (Eds.) (1995). *Proceedings of the First International Conference on Knowledge Discovery and Data Mining.* Menlo Park, CA: AAAI Press.

- Fayyad, U. M., Smyth, P., Weir, N., Djorgovski, S. (1995). "Automated analysis and exploration of image databases: Results, progress, and challenges". *Journal of Intelligent Information Systems,* 4, 1-19.

- Fisher R. A. (1936). "The Use of Multiple Measurements in Taxonomic Problems". *Annual Eugenics*, 7, 179-188.

- Fogel, D. B. (1988). "An evolutionary approach to the traveling salesman problem". *Biological Cybernetics*, 60, 139-144.

- Fogel, D. B. (1993). "Applying evolutionary programming to selected travelling salesman problems". *Cybernetics and Systems*, 24, 27-36.

- Forrest, S. (1993). "Genetic Algorithms: Principles of Natural Selection Applied to Computation". *Science,* Vol. 261, No. 5123, pp. 872-878.

- Gorges-Schleuter, M. (1989). "ASPARAGOS: An Asynchronous Parallel Genetic Optimization Strategy". *Proceedings of the Third Int. Conf. on Genetic Algorithms* (ICGA'89), George Mason University, Fairfax, VA, pp. 422-427.

- Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley.

- Goldberg, D. E., Lingle, Jr. R. (1985). "Alleles, loci and the traveling salesman problem". *Proceedings of the First International Conference on Genetic Algorithms and Their Applications*, 154-159.

- Goldberg, D. E., Richardson, J. T. (1987). "Genetic algorithms with sharing for multimodal function optimization". *Genetic Algorithms and their Applications: Proceedings of the Second International Conference on Genetic Algorithms and Their Applications*, 41-49.

- Grefenstette, J., Gopal, R., Rosmaita, B., Van Gucht, D. (1985). "Genetic algorithms for the traveling salesman problem". *Proceedings of the First International Conference on Genetic Algorithms and Their Applications*, 160-165.

- Holland, J. (1992). "Genetic algorithms". *Scientific American*, p. 66-72.

- Larrañaga, P., Kuijpers, C., Murga, R., Inza, I., Dizdarevich, S. (1999) "Evolutionary algorithms for the travelling salesman problem: A review of representations and operators". *Artificial Intelligence Review*, 13, 129–170.

- Lee, K. (1989). *Automatic speech recognition: The development of the Sphinx system.* Boston: Kluwer Academic Publishers.

- Lerman, I. C. (1981). *Classification et analyse ordinale des données.* Dunod, Paris.

- Michalewicz, Z. (1992). Genetic *Algorithms + Data Structures = Evolution Programs*, Springer-Verlag, Berlin Heidelberg.

- Mitchell, M.(1996). *An introduction to Genetic Algorithms*. MIT Press, 1996.

- Mulhenbein, H., Gorges-Schleuter, M. and Kramer, O. (1987). "New Solutions to the Mapping Problem of Parallel Systems- The Evolution Approach". *Parallel Computing 4*, pp. 269-279.

- Mulhenbein, H., Gorges-Schleuter, M. and Kramer, O. (1988). "Evolution Algorithms in Combinatorial Optimization". *Parallel Computing 7*, pp. 65-85.

- Mulhenbein, H. (1991). "Evolution in Time and Space – The Parallel Genetic Algorithm", *Foundations of Genetic Algorithms*, G.J.E. Rawlins Eds, Morgan Kaufmann, pp. 316-337.

- Oliver, I. M., Smith, D. J., Holland, J. R. C. (1987). "A study of permutation crossover operators on the TSP". *Genetic Algorithms and Their Applications: Proceedings of the Second International Conference*, 224-230.

- Pomerleau, D. A. *ALVINN: An autonomous land vehicle in a neural network.* (Technical Report CMU-CS-89-107). Pittsburgh, PA: Carnegie Mellon University.

- Syswerda, G. (1991). "Schedule optimization using genetic algorithms". Davis, L. (ed.). *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, New York, 332-349.

- Tesauro, G. (1992). "Practical issues in temporal difference learning". *Machine Learning,* 8, 257.

- Tesauro, G. (1995). "Temporal difference learning and TD-gammon". *Communications of the ACM,* 38(3), 58-68.

- Waibel, A., Hanazawa, T., Hinton, G., Shikano, K., & Lang, K. (1989). "Phoneme recognition using time-delay neural networks". *IEEE Transactions on Acoustics, Speech and Signal Processing,* 37(3), 328-339.

- Whitley, D., Starkweather, T., Fuquay, D. (1989). "Scheduling problems and travelling salesman: The genetic edge recombination operator". *Proceedings on the Third International Conference on Genetic Algorithms*, 133-140.

- Whitley, D., Starkweather, T. (1990). "Genitor II: A distributed genetic algorithm". *Journal of Experimental and Theorical Artificial Intelligence*, 2, 189-214.

- Whitley, D., Starkweather, T., Shaner, D. (1991). "The traveling salesman and sequence scheduling: Quality solutions using genetic edge recombination", Davis, L. (ed.) *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, New York, 350-372.