Algoritmos Genéticos. Aplicación al Juego de las N Reinas.

Juan Carlos Pozas Bustos
NIA: 100025154
Univ.Carlos III de Madrid
Ing.Telecomunicación
España
100025154@alumnos.uc3m.es

Nieves Vázquez Vázquez
NIA: 100025194
Univ.Carlos III de Madrid
Ing.Telecomunicación
España
100025194@alumnos.uc3m.es

Términos generales

En el presente documento estudiaremos la historia, el desarrollo, las ventajas y limitaciones de los algoritmos genéticos así como una de sus posibles aplicaciones en la resolución de problemas.

1. INTRODUCCIÓN

Durante el curso de la historia, los seres humanos hemos construido gradualmente un gran edificio de conocimiento que nos permite predecir el tiempo, los movimientos de los planetas, los eclipses solares y lunares, el desarrollo de enfermedades, la subida y la caída del desarrollo económico y un panorama extenso de otros fenómenos naturales, sociales, y culturales.

Hemos desarrollado medios cada vez más complejos para controlar muchos aspectos de nuestras vidas y de nuestras interacciones con la naturaleza.

Esta revolución actual está aumentando nuestra capacidad de predecir y de controlar la naturaleza en términos que jamás fueron concebidos. Para muchos, los mejores logros de esta revolución serán la creación -en forma de programas- de nuevas especies de seres inteligentes, o incluso de nuevas formas de vida.

Los objetivos de crear inteligencia artificial y vida artificial se remontan a los orígenes de la era del ordenador. Alan Turing, Jon von Neumann, Norbert Wiener, etc. estaban, no sólo interesados en la electrónica, sino también en la idea de desarrollar programas de ordenador con la inteligencia y con la capacidad adaptativa de aprender y controlar los entornos. Así que desde sus primeros pasos, los ordenadores estaban también concebidos para modelar el cerebro, imitar el aprendizaje humano y simular la evolución biológica. Estos tres campos han dado lugar a las redes neuronales, el aprendizaje máquina y a la computación evolutiva, cuyo ejemplo más predominante son los algoritmos genéticos.

Los algoritmos genéticos están inspirados en la naturaleza, en el fenómeno de la evolución. La evolución la consideramos como la causa de los cambios en el contenido genético de una población.

Un tema polémico, con opiniones variadas dependiendo de si se trata de informáticos evolutivos o de biólogos o geneticistas, es si la evolución optimiza o no.

Según los **informáticos evolutivos**, la evolución optimiza, puesto que va creando seres cada vez más perfectos, cuya cumbre es el hombre; además, indicios de esta optimización se encuentran en el organismo de los animales, desde el tamaño y tasa de ramificación de las arterias, diseñada para maximizar flujo, hasta el metabolismo, que optimiza la cantidad de energía extraída de los alimentos.

Sin embargo, **los geneticistas y biólogos evolutivos** afirman que la evolución no optimiza, sino que adapta y optimiza localmente en el espacio y el tiempo; evolución no significa progreso. Un organismo más evolucionado puede estar en desventaja competitiva con uno de sus antepasados, si se colocan en el ambiente del último.

2. BREVE 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 por biólogos evolutivos que buscaban 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 hecho no tardaría en llegar.

John Holland se preguntaba cómo lograba la naturaleza, crear seres cada vez más perfectos. No sabía la respuesta, pero tenía una cierta idea de como 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.

Fue a principios de los 60, 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 ese 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 fin determinado.

En esa universidad, Holland impartía un curso titulado *Teoría de 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, ingeniero industrial, fue uno de los primeros que trató de aplicar los algoritmos genéticos a problemas industriales. Aunque Holland trató de disuadirle, 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 como para celebrar la primera conferencia en 1985, ICGA´85.

3. ¿QUÉ ES UN ALGORITMO GENÉTICO?

Los Algoritmos Genéticos (GA) pueden verse como una familia de procedimientos de búsqueda adaptivos. Están basados en modelos de cambio genético en una población de individuos. Un punto clave de estos modelos, es que el proceso de adaptación no se hace cambiando incrementalmente una sola estructura, sino manteniendo una población de estructuras a partir de las cuales se generan nuevas estructuras usando los operadores genéticos. Esto es:

- Noción darwiniana de aptitud (***fitness***) que influye en generaciones futuras.

- Apareamiento que produce descendientes en generaciones futuras.

- Operadores genéticos que determinan la configuración genética de los descendientes (tomada de los padres).

Un algoritmo genético es una técnica de programación que imita a la evolución biológica como estrategia para resolver problemas. Dado un problema específico a resolver, la entrada del AG es un conjunto de soluciones potenciales a ese problema, codificadas de alguna manera, y una métrica llamada **función de aptitud** que permite evaluar cuantitativamente a cada solución candidata. Estas candidatas pueden ser soluciones que ya se sabe que funcionan, con el objetivo de que el AG las mejore, pero se suelen generar aleatoriamente.

Luego el AG evalúa cada candidata de acuerdo con la función de aptitud. En un conjunto de soluciones candidatas generadas aleatoriamente, por supuesto, la mayoría no funcionarán en absoluto, y serán eliminadas. Sin embargo, por puro azar, unas pocas pueden ser prometedoras, pueden mostrar actividad, aunque sólo sea actividad débil e imperfecta, hacia la solución del problema.

Estas candidatas prometedoras se conservan y se les permite reproducirse. Se realizan múltiples copias de ellas, pero las copias no son perfectas; se introducen cambios aleatorios durante el proceso de copia. Luego, esta descendencia digital prosigue con la siguiente generación, formando un nuevo acervo de soluciones candidatas, y son sometidas a una ronda de evaluación de aptitud. Las candidatas que han empeorado o no han mejorado con los cambios en su código son eliminadas de nuevo; pero, nuevamente, por puro azar, las variaciones aleatorias introducidas en la población pueden haber mejorado a algunos individuos, convirtiéndolos en mejores soluciones del problema, más completas o más eficientes. De nuevo, se seleccionan y copian estos individuos vencedores hacia la siguiente generación con cambios aleatorios, y el proceso se repite.

Las expectativas son que la aptitud media de la población se incrementará en cada ronda y, por tanto, repitiendo este proceso cientos o miles de rondas, pueden descubrirse soluciones muy buenas del problema.

Aunque a algunos les puede parecer asombroso y antiintuitivo, los algoritmos genéticos han demostrado ser una estrategia enormemente poderosa y exitosa para resolver problemas, demostrando de manera espectacular el poder de los principios evolutivos.

Se han utilizado algoritmos genéticos en una amplia variedad de campos para desarrollar soluciones a problemas tan difíciles o más que los abordados por los diseñadores humanos. Y las soluciones que consiguen son a menudo más eficientes y elegantes.

4. REPRESENTACIÓN DE ALGORITMOS GENÉTICOS

Antes de que un algoritmo genético pueda ponerse a trabajar en un problema, se necesita un método para codificar las soluciones potenciales del problema de forma que una computadora pueda procesarlas.

Un enfoque común es codificar las soluciones como cadenas binarias: secuencias de 1's y 0's, donde el dígito de cada posición representa el valor de algún aspecto de la solución.

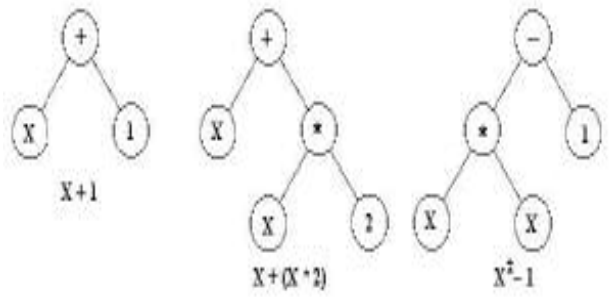
Otro método similar consiste en codificar las soluciones como cadenas de enteros o números decimales. Este método permite una mayor precisión y complejidad que el anterior y a menudo está intuitivamente más cerca del espacio de problemas. Los algoritmos genéticos para entrenar a las redes neuronales utilizan a menudo este método de codificación.

Un tercer método consiste en representar a los individuos de un AG como cadenas de letras, donde cada letra, de nuevo, representa un aspecto específico de la solución. También se usa este tipo de codificación para entrenar redes neuronales.

La virtud de estos tres métodos es que facilitan la definición de operadores que causen los cambios aleatorios en las candidatas seleccionadas: cambiar un 0 por un 1 o viceversa, sumar o restar al valor de un número una cantidad elegida al azar, o cambiar una letra por otra.

Otra estrategia, desarrollada principalmente por John Koza, de la Universidad de Stanford, y denominada programación

genética, representa a los programas como estructuras de datos ramificadas llamadas árboles. En este método, los cambios aleatorios pueden generarse cambiado el operador o alterando el valor de un cierto nodo del árbol, o sustituyendo un subárbol por otro. En la figura se muestran tres ejemplos de árboles de programa, con las expresiones matemáticas que representa cada uno:



Es importante señalar que los algoritmos evolutivos no necesitan representar las soluciones candidatas como cadenas de datos de una longitud fija. Algunos las representan de esta manera, pero otros no. Por ejemplo, la "codificación gramatical" mencionada previamente, puede escalarse eficientemente para crear redes neuronales grandes y complejas, y los árboles de programación genética pueden crecer arbitrariamente tanto como sea necesario para resolver cualquier problema que se les pida.

5. ALGORITMO GENÉTICO SIMPLE

Se estructura en los siguientes pasos:

a) Primero, se genera aleatoriamente la población inicial, que está constituida por un conjunto de cromosomas, que representan las posibles soluciones del problema. En caso de no hacerlo aleatoriamente, es importante garantizar que dentro de la población inicial, se tenga la diversidad estructural de estas soluciones para tener una representación de la mayor parte de la población posible o al menos evitar la convergencia prematura.

b) A cada uno de los cromosomas de esta población se aplicará la función de aptitud para saber lo "buena" que es la solución que se está codificando.

c) Después de saber la aptitud de cada cromosoma se procede a elegir los cromosomas que serán cruzados en la siguiente generación.

d) Los cromosomas con mejor aptitud tienen mayor probabilidad de ser seleccionados.

e) Se aplica un operador genético: mutación, cruce, evaluación…

f) El AG se deberá detener cuando se alcance la solución óptima, pero ésta generalmente se desconoce, por lo que se deben utilizar otros criterios de detención. Normalmente se usan dos: correr el AG un número máximo de iteraciones (generaciones) ó detenerlo cuando no haya cambios en la población.

6. OPERADORES GENÉTICOS

6.1 Selección

Un algoritmo genético puede utilizar muchas técnicas diferentes para seleccionar a los individuos que deben copiarse hacia la siguiente generación. A continuación se listan algunos de los más comunes. Algunos de estos métodos son mutuamente exclusivos, pero otros pueden utilizarse en combinación, algo que se hace a menudo.

- **Selección elitista**: se garantiza la selección de los miembros más aptos de cada generación. (La mayoría de los AGs no utilizan elitismo puro, sino que usan una forma modificada por la que el individuo mejor, o algunos de los mejores, son copiados hacia la siguiente generación en caso de que no surja nada mejor).

- **Selección proporcional a la aptitud**: los individuos más aptos tienen más probabilidad de ser seleccionados, pero no la certeza.

- **Selección por rueda de ruleta**: una forma de selección proporcional a la aptitud en la que la probabilidad de que un individuo sea seleccionado es proporcional a la diferencia entre su aptitud y la de sus competidores. (Conceptualmente, esto puede representarse como un juego de ruleta -cada individuo obtiene una sección de la ruleta, pero los más aptos obtienen secciones mayores que las de los menos aptos. Luego la ruleta se hace girar, y en cada vez se elige al individuo que "posea" la sección en la que se pare la ruleta).

- **Selección escalada**: al incrementarse la aptitud media de la población, la fuerza de la presión selectiva también aumenta y la función de aptitud se hace más discriminadora. Este método puede ser útil para seleccionar más tarde, cuando todos los individuos tengan una aptitud relativamente alta y sólo les distingan pequeñas diferencias en la aptitud.

- **Selección por torneo**: se eligen subgrupos de individuos de la población, y los miembros de cada subgrupo compiten entre ellos. Sólo se elige a un individuo de cada subgrupo para la reproducción.

- **Selección por rango**: a cada individuo de la población se le asigna un rango numérico basado en su aptitud, y la selección se basa en este ranking, en lugar de las diferencias absolutas en aptitud. La ventaja de este método es que puede evitar que individuos muy aptos ganen dominancia al principio a expensas de los menos aptos, lo que reduciría la diversidad genética de la población y podría obstaculizar la búsqueda de una solución aceptable.

- **Selección generacional**: la descendencia de los individuos seleccionados en cada generación se convierte en toda la siguiente generación. No se conservan individuos entre las generaciones.

- **Selección por estado estacionario**: la descendencia de los individuos seleccionados en cada generación vuelven al acervo genético preexistente, reemplazando a algunos de los miembros menos aptos de la siguiente generación. Se conservan algunos individuos entre generaciones.

- **Selección jerárquica**: los individuos atraviesan múltiples rondas de selección en cada generación. Las evaluaciones de los primeros niveles son más rápidas y menos discriminatorias, mientras que los que sobreviven hasta niveles más altos son evaluados más rigurosamente. La ventaja de este método es que reduce el tiempo total de cálculo al utilizar una evaluación más rápida y menos selectiva para eliminar a la mayoría de los individuos que se muestran poco o nada prometedores, y sometiendo a una evaluación de aptitud más rigurosa y computacionalmente más costosa sólo a los que sobreviven a esta prueba inicial.

6.2 Mutación

En la evolución, una mutación es un suceso bastante poco común (sucede aproximadamente una de cada mil replicaciones). En la mayoría de los casos las mutaciones son letales, pero en promedio, contribuyen a la diversidad genética de la especie.

Una vez establecida la frecuencia de mutación, por ejemplo, uno por mil, se examina cada bit de cada cadena cuando se vaya a crear la nueva criatura a partir de sus padres (normalmente se hace de forma simultánea al crossover). Si un número generado aleatoriamente está por debajo de esa probabilidad, se cambiará el bit (es decir, de 0 a 1 o de 1 a 0). Si no, se dejará como está. Dependiendo del número de individuos que haya y del número de bits por individuo, puede resultar que las mutaciones sean extremadamente raras en una sola generación.

No conviene abusar de la mutación. Es cierto que es un mecanismo generador de diversidad, y, por tanto, la solución cuando un algoritmo genético está estancado, pero también es cierto que reduce el algoritmo genético a una búsqueda aleatoria. Siempre es más conveniente usar otros mecanismos de generación de diversidad, como aumentar el tamaño de la población, o garantizar la aleatoriedad de la población inicial.

6.3 Cruce (Crossover)

Consiste en el intercambio de material genético entre dos cromosomas. El *crossover* es el principal operador genético, hasta el punto que se puede decir que no es un algoritmo genético si no tiene *crossover*, y, sin embargo, puede serlo perfectamente sin mutación, según descubrió Holland. El *teorema de los esquemas* confía en él para hallar la mejor solución a un problema, combinando soluciones parciales.

El teorema de los esquemas proporciona el fundamento teórico de por qué los AG pueden resolver diversos problemas. En su análisis se considera el proceso de selección y los operadores de cruce y mutación. Se basa en la noción de *bloques de construcción*. Una buena solución a un problema está constituida por unos buenos bloques, igual que una buena máquina está hecha por buenas piezas. El crossover es el encargado de mezclar bloques buenos que se encuentren en los diversos progenitores, y que serán los que den a los mismos una buena puntuación. La presión selectiva se encarga de que sólo los buenos bloques se perpetúen, y poco a poco vayan formando una buena solución. El *teorema de los esquemas* viene a decir que la cantidad de *buenos bloques* se va incrementando con el tiempo de ejecución de un algoritmo genético, y es el resultado teórico más importante en algoritmos genéticos.

Un esquema se construye utilizando un nuevo símbolo (*) para representar un comodín que puede aparear ambos valores (0 o 1). Ejm., el esquema 11*00* representa las cadenas: 111001, 111000, 110001, 110000.

El orden de un esquema es el número de elementos que no son "*" dentro del esquema. La longitud que define a un esquema es la distancia entre la primera posición fija y la última posición fija.

El teorema dice que si existen $N(S,t)$ instancias de un esquema S en una población en el instante t, en el siguiente tiempo el valor esperado de instancias en la nueva población esta acotado por:

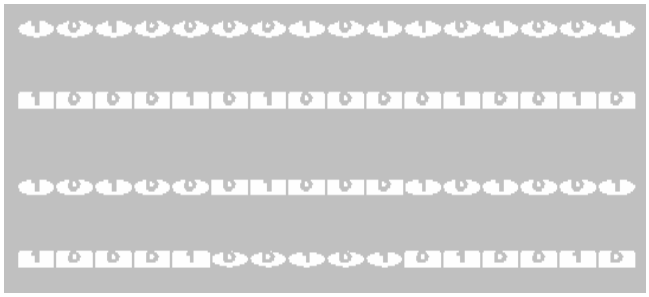
$$E[N(S,t+1)] \geq \frac{F(S,t)}{\overline{F}(t)} N(S,t)[1-\epsilon(S,t)]$$

donde $F(S,t)$ es la aptitud del esquema S, $\overline{F}(t)$ es la aptitud promedio de la población, y $\in(S,t)$ es un término que refleja el potencial del algoritmo genético de destruir instancias del esquema S.

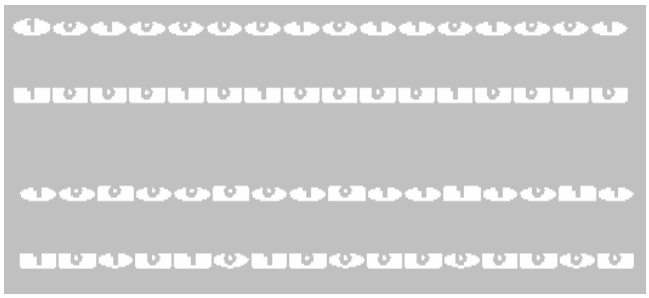
Para aplicar el crossover, entrecruzamiento o recombinación, se escogen aleatoriamente dos miembros de la población. No pasa nada si se emparejan dos descendientes de los mismos padres; ello garantiza la perpetuación de un individuo con buena puntuación. Sin embargo, si esto sucede demasiado a menudo, puede crear problemas: toda la población puede aparecer dominada por los descendientes de algún gen, que, además, puede tener caracteres no deseados. Esto se suele denominar en otros métodos de optimización *atranque en un mínimo local*, y es uno de los principales problemas con los que se enfrentan los que aplican algoritmos genéticos.

El intercambio genético se puede llevar a cabo de muchas formas, pero hay dos grupos principales:

- *Crossover n-puntos*: los dos cromosomas se cortan por n puntos, y el material genético situado entre ellos se intercambia. Lo más habitual es un crossover de un punto o de dos puntos.



- *Crossover uniforme*: se genera un patrón aleatorio de 1's y 0's, y se intercambian los bits de los dos cromosomas que coincidan donde hay un 1 en el patrón. O bien se genera un número aleatorio para cada bit, y si supera una determinada probabilidad se intercambia ese bit entre los dos cromosomas.



- *Crossover especializado*: en algunos problemas, aplicar aleatoriamente el crossover da lugar a cromosomas que codifican soluciones inválidas; en este caso hay que aplicar el crossover de forma que genere siempre soluciones válidas.

En toda ejecución de un algoritmo genético hay que decidir con qué frecuencia se va a aplicar cada uno de los operadores genéticos; en algunos casos, como en la mutación o el crossover uniforme, se debe de añadir algún parámetro adicional, que indique con qué frecuencia se va a aplicar dentro de cada gen del cromosoma. La frecuencia de aplicación de cada operador estará en función del problema; teniendo en cuenta los efectos de cada operador, tendrá que aplicarse con cierta frecuencia o no. Generalmente, la mutación y otros operadores que generen diversidad se suele aplicar con poca frecuencia; la recombinación se suele aplicar con frecuencia alta.

En general, la frecuencia de los operadores no varía durante la ejecución del algoritmo, pero hay que tener en cuenta que cada operador es más efectivo en un momento de la ejecución. Por ejemplo, al principio, en la fase denominada de *exploración*, los más eficaces son la mutación y la recombinación; posteriormente, cuando la población ha convergido en parte, la recombinación no es útil, pues se está trabajando con individuos bastante similares, y es poca la información que se intercambia. Sin embargo, si se produce un estancamiento, la mutación tampoco es útil, pues está reduciendo el algoritmo genético a una búsqueda aleatoria; y hay que aplicar otros operadores. En todo caso, se pueden usar operadores especializados.

7. OTROS OPERADORES

7.1 Cromosomas de longitud variable

Hasta ahora se han descrito cromosomas de longitud fija, donde se conoce de antemano el número de parámetros de un problema. Pero hay problemas en los que esto no sucede. En estos casos, necesitamos dos operadores más: *añadir* y *eliminar*. Estos operadores se utilizan para añadir un gen, o eliminar un gen del cromosoma. La forma más habitual de añadir es *duplicar* uno ya existente, el cual sufre mutación y se añade al lado del anterior. En este caso, los operadores del algoritmo genético simple (selección, mutación, crossover) funcionarán de la forma habitual, salvo, claro está, que sólo se hará crossover en la zona del cromosoma de menor longitud.

Estos operadores permiten, además, crear un *algoritmo genético de dos niveles*: a nivel de cromosoma y a nivel de gen. Supongamos que, en un problema de clasificación, hay un gen por clase. Se puede asignar una puntuación a cada gen en función del número de muestras que haya clasificado correctamente. Al aplicar estos operadores, se duplicarán los alelos con mayor puntuación, y se eliminarán aquellos que hayan obtenido menor puntuación, o cuya puntuación sea nula.

7.2 Operadores de nicho (ecológico)

Estos operadores están encaminados a mantener la diversidad genética de la población, de forma que cromosomas similares sustituyan sólo a cromosomas similares, y son especialmente útiles en problemas con muchas soluciones; un algoritmo genético con estos operadores es capaz de hallar todos los máximos, dedicándose cada especie a un máximo. Más que operadores genéticos, son formas de enfocar la selección y la evaluación de la población.

Uno de las formas de llevar esto a cabo ya ha sido nombrada, la introducción del *crowding (apiñamiento)*. Otra forma es introducir una función de compartición o *sharing*, que indica cuán similar es un cromosoma al resto de la población. La puntuación de cada individuo se dividirá por esta función de compartición, de forma que se facilita la diversidad genética y la aparición de individuos diferentes.

7.3 Operadores especializados

En una serie de problemas hay que restringir las nuevas soluciones generadas por los operadores genéticos, pues no todas las soluciones generadas van a ser válidas, sobre todo en los problemas con restricciones. Por ello, se aplican operadores que mantengan la estructura del problema. Otros operadores son simplemente generadores de diversidad, pero la generan de una forma determinada:

- **Zap**: en vez de cambiar un solo bit de un cromosoma, cambia un gen completo de un cromosoma.

- **Creep:** este operador aumenta o disminuye en 1 el valor de un gen; sirve para cambiar suavemente y de forma controlada los valores de los genes.

- **Transposición:** similar al crossover y a la recombinación genética, pero dentro de un solo cromosoma; dos genes intercambian sus valores, sin afectar al resto del cromosoma. Similar a este es el operador de **eliminación-reinserción**, en el que un gen cambia de posición con respecto a los demás.

8. VENTAJAS DE LOS ALGORITMOS GENÉTICOS

- El primer y más importante punto es que los algoritmos genéticos son intrínsecamente paralelos. La mayoría de los otros algoritmos son en serie y sólo pueden explorar el espacio de soluciones hacia una solución en una dirección al mismo tiempo, y si la solución que descubren resulta subóptima, no se puede hacer otra cosa que abandonar todo el trabajo hecho y empezar de nuevo. Sin embargo, ya que los algoritmos genéticos tienen descendencia múltiple, pueden explorar el espacio de soluciones en múltiples direcciones a la vez. Si un camino resulta ser un callejón sin salida, pueden eliminarlo fácilmente y continuar el trabajo en avenidas más prometedoras, dándoles una mayor probabilidad en cada ejecución de encontrar la solución.

  Sin embargo, la ventaja del paralelismo va más allá de esto. Todas las cadenas binarias de 8 dígitos forman un espacio de búsqueda, que puede representarse como ******** (donde * significa ``o 0 o 1''). La cadena 01101010 es un miembro de este espacio. Sin embargo, también es un miembro del espacio 0*******, del espacio 01******, del espacio 0******0, del espacio 0*1*1*1*, del espacio 10*01**0, etcétera. Evaluando la aptitud de esta cadena particular, un algoritmo genético estaría sondeando cada uno de los espacios a los que pertenece. Tras muchas evaluaciones, iría obteniendo un valor cada vez más preciso de la aptitud media de cada uno de estos espacios, cada uno de los cuales contiene muchos miembros. Por tanto, un algoritmo genético que evalúe explícitamente un número pequeño de individuos está evaluando implícitamente un grupo de individuos mucho más grande.

  De la misma manera, el algoritmo genético puede dirigirse hacia el espacio con los individuos más aptos y encontrar el mejor de ese grupo. En el contexto de los algoritmos evolutivos, esto se conoce como teorema del esquema, y es la ventaja principal de los algoritmos genéticos sobre otros métodos de resolución de problemas.

- Debido al paralelismo que les permite evaluar implícitamente muchos esquemas a la vez, los algoritmos genéticos funcionan particularmente bien resolviendo problemas cuyo espacio de soluciones potenciales es realmente grande. La mayoría de los problemas que caen en esta categoría se conocen como “no lineales”. En un problema lineal, la aptitud de cada componente es independiente, por lo que cualquier mejora en alguna parte dará como resultado una mejora en el sistema completo. Sin embargo la no linealidad es la norma en la vida real, donde cambiar un componente puede tener efectos en cadena en todo el sistema, y donde cambios múltiples que, individualmente, son perjudiciales, en combinación pueden conducir hacia mejoras en la aptitud mucho mayores. La no linealidad produce una explosión combinatoria, sobre la que, gracias al paralelismo implícito de los AGs se les permite superar estos problemas ante un enorme número de posibilidades, encontrando con éxito resultados óptimos o muy buenos en un corto periodo de tiempo.

- Otra ventaja notable de los algoritmos genéticos es que se desenvuelven bien en problemas donde la función de aptitud es discontinua, ruidosa, cambia con el tiempo, o tiene muchos óptimos locales. La mayoría de los problemas prácticos tienen un espacio de soluciones enorme, imposible de explorar exhaustivamente; el reto se convierte entonces en cómo evitar los óptimos locales.

  Los algoritmos evolutivos han demostrado su efectividad al escapar de dichos óptimos locales y descubrir el óptimo global incluso en paisajes adaptativos muy escabrosos y complejos. Los cuatro componentes principales de los algoritmos genéticos -paralelismo, selección, mutación y cruzamiento- trabajan juntos para conseguir esto.

- Otra área en el que destacan los algoritmos genéticos es en su habilidad para manipular muchos parámetros simultáneamente. Su uso del paralelismo les permite producir múltiples soluciones, igualmente buenas, al mismo problema, donde posiblemente una solución candidata optimiza un parámetro y otra candidata optimiza uno distinto, y luego un supervisor humano puede seleccionar una de esas candidatas para su utilización.

- Finalmente, una de las cualidades de los algoritmos genéticos que, a primera vista, puede parecer un desastre, resulta ser una de sus ventajas: los algoritmos genéticos no saben nada de los problemas que deben resolver. En lugar de utilizar información específica conocida a priori para guiar cada paso realizan cambios aleatorios en sus soluciones candidatas y luego utilizan la función de aptitud para determinar si esos cambios producen una mejora.

  Por ello, todos los caminos de búsqueda posibles están abiertos sin perder así cualquier solución novedosa que pueda existir. No obstante, cualquier técnica que dependa de conocimiento previo fracasará ante dicho planteamiento.

9. LIMITACIONES DE LOS ALGORITMOS GENÉTICOS

Los AGs tienen ciertas limitaciones; sin embargo, se demostrará que todas ellas pueden superarse y que ninguna afecta a la validez de la evolución biológica.

- La primera y más importante consideración al crear un algoritmo genético es definir una representación del problema. El lenguaje utilizado para especificar soluciones candidatas debe ser robusto; es decir, debe ser capaz de tolerar cambios aleatorios que no produzcan constantemente errores fatales o resultados sin sentido.

Hay dos maneras principales para conseguir esto. La primera, utilizada por la mayoría de los algoritmos genéticos, es definir a los individuos como listas de números -binarios, enteros o reales- donde cada número representa algún aspecto de la solución candidata. Si los individuos son cadenas binarias, un 0 o 1 podría significar la ausencia o presencia de una cierta característica. Así, la mutación implica cambiar estos números, cambiar bits o sumar o restar valores aleatorios. En este caso, el propio código del programa no cambia; el código es lo que dirige la simulación y hace un seguimiento de los individuos, evaluando sus aptitudes y quizá asegurando que sólo se producen valores realistas y posibles para el problema dado.

En otro método, la programación genética, el propio código del programa *sí* cambia. Sería el casoya descrito en el que a los individuos se les representa como árboles de código ejecutables que pueden mutar cambiando o intercambiando subárboles.

Ambos métodos producen representaciones robustas ante la mutación, y pueden representar muchos tipos diferentes de problemas.

- El problema de cómo escribir la función de aptitud debe considerarse cuidadosamente para que se pueda alcanzar una mayor aptitud y verdaderamente signifique una solución mejor para el problema dado. Si se elige mal una función de aptitud o se define de manera inexacta, puede que el algoritmo genético sea incapaz de encontrar una solución al problema, o puede acabar resolviendo el problema equivocado.

- Además de elegir bien la función de aptitud, también deben elegirse cuidadosamente los otros parámetros de un AG -el tamaño de la población, el ritmo de mutación y cruzamiento, el tipo y fuerza de la selección. Si el tamaño de la población es demasiado pequeño, puede que el algoritmo genético no explore suficientemente el espacio de soluciones para encontrar buenas soluciones consistentemente. Si el ritmo de cambio genético es demasiado alto o el sistema de selección se escoge inadecuadamente, puede alterarse el desarrollo de esquemas beneficiosos y la población puede entrar en catástrofe de errores, al cambiar demasiado rápido para que la selección llegue a producir convergencia.

- Otro problema con el que los algoritmos genéticos tienen dificultades es con las funciones de aptitud "engañosas" en las que la situación de los puntos mejorados ofrecen información engañosa sobre dónde se encuentra probablemente el óptimo global. Sin embargo puede funcionar casi igual de bien alcanzando la cima de un óptimo local alto y, para la mayoría de las situaciones, eso será suficiente, incluso aunque el óptimo global no pueda alcanzarse fácilmente desde ese punto. A demás la situación de las mejoras locales suelen proporcionar alguna información sobre la situación del óptimo global.

- Otro problema muy conocido que puede surgir es la convergencia prematura. Si un individuo que es más apto que la mayoría de sus competidores emerge muy pronto en el curso de la ejecución, se puede reproducir tan abundantemente que merme la diversidad de la población demasiado pronto, provocando que el algoritmo converja hacia el óptimo local que representa ese individuo. Esto es un problema especialmente común en las poblaciones pequeñas.

  Los métodos más comunes implementados para solucionar este problema implican controlar la fuerza selectiva, para no proporcionar tanta ventaja a los individuos excesivamente aptos. La selección escalada, por rango y por torneo, discutidas anteriormente, son tres de los métodos principales para conseguir esto.

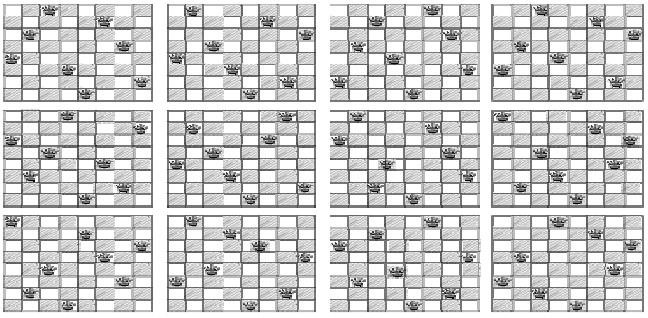
- Finalmente, varios investigadores (Holland, Forrest, Haupt y Haupt) aconsejan no utilizar algoritmos genéticos en problemas resolubles de manera analítica. No es que los algoritmos genéticos no puedan encontrar soluciones buenas para estos problemas; simplemente es que los métodos analíticos tradicionales consumen mucho menos tiempo y potencia computacional que los AGs y, a diferencia de los AGs, a menudo está demostrado matemáticamente que ofrecen la única solución exacta.

10. APLICACIÓN AL JUEGO DE LAS N REINAS

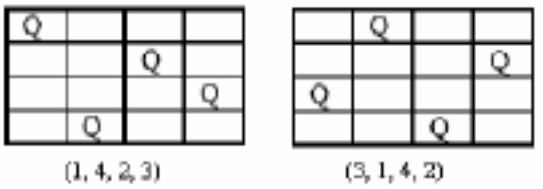
Los problemas que no tienen soluciones deterministas que se ejecutan en tiempo polinomial se denominan problemas de tipo NP. Debido a su elevado complejidad ($O(2^n)$ o O(n!)), deben ser resueltos en una cantidad de tiempo razonable empleando métodos heurísticos. Los algoritmos genéticos son métodos de este estilo muy poderosos, capaces de realizar búsquedas en grandes espacios de soluciones de forma muy eficiente. Sin embargo, debido al número elevado de cálculos que han de realizarse, algún tipo de paralelismo es necesario.

El problema original de ocho reinas consistía en intentar encontrar una forma de colocar a ocho reinas en un tablero de ajedrez de modo que dos reinas no se atacaran la una a la otra. Es decir, que en un tablero de 8 por 8, ninguna de las reinas comparta una fila, columna o diagonal.

Existen 92 soluciones a este problema, de las cuales 12 tienen un patrón distinto. Cada una de las 92 soluciones puede ser transformada en una de estos 12 patrones, utilizando rotaciones y reflexiones. La disposición de las 8 reinas en las 12 soluciones básicas se muestra en la siguiente figura:



Podemos generalizar este problema para poder disponer n reinas en un tablero de tamaño n x n sin que ninguna sea atacada. Como cada reina debe estar en una fila o columna diferente, podemos asumir que la reina *i* se sitúa en la columna *i*. Todas las soluciones al problema de las n reinas se pueden representar como n tuplas (q_1, q_2,..., qn), que son permutaciones de una tupla n (1, 2,…, n). La posición de un número en la tupla representa la posición de la columna de la reina, mientras que su valor representa la posición de la fila de la reina. Utilizando esta representación, el espacio de soluciones, donde el conflicto entre fila y columna ya está solucionado, debe ser encontrado para poder eliminar los conflictos de la diagonal. La complejidad de este problema es O(n!). A continuación se muestran dos ejemplos de posibles soluciones para el problema de 4 reinas:



El problema de determinar una función de aptitud o fitness es igual que para cualquier problema de combinatoria: la solución es tanto correcta como incorrecta. Por tanto, una función de fitness debe ser capaz de determinar lo cerca que una solución incorrecta está de una correcta. Como la representación explicada previamente elimina el conflicto entre filas y columnas, las soluciones incorrectas son aquellas en las que las reinas se atacan unas a otras diagonalmente. Una función de fitness puede ser implementada para contar el número de conflictos que hay en la diagonal: cuantos más conflictos haya, más errónea será la solución. Para una solución correcta, la función de fitness deberá devolver como resultado un 0.

En la implementación realizada se ha elegido la siguiente función de fitness: considerando una n-tupla (q_1,..., q_i,..., qj,..., qn), las reinas i y j compartirán la diagonal si:

$$i - q_i = j - q_j$$

o

$$i + q_i = j + q_j$$

, que se reduce a:

$$\|q_i - q_j\| = \|i - j\|$$

A continuación se adjunta la función principal del algoritmo genético implementado en MATLAB:

```
function [ fitnessMedio mejorFitness mejorIndividuo ] = GAnReinas( nReinas, NIND, MAXGEN, porcNewInd,pMutacion, funSeleccion)
% GAnReinas Resuelve el problema de las n-reinas con un algoritmo genético
%
% INPUT:
%   nReinas: número de reinas
%   NIND: número de individuos de cada generación
%   MAXGEN: número máximo de generaciones
%   porcNewInd: porcentaje de nuevos individuos en la siguiente %generación
%   porcMutacion: porcentaje de mutación
%   funSeleccion: operador de selección (Ruleta o Torneo)
%
% OUTPUT:
%   fitnessMedio: vector con el fitness medio de cada generación
%   mejorFitness: vector con el mejor fitness de cada generación
% mejorIndividuo: matriz que contiene la representación del mejor individuo de cada generación
% Representación:
% Se utiliza un vector con tantas posiciones como columnas tiene % el tablero. Cada valor del vector indica la fila en la que se %encuentra la reina en esa columna. No se permiten valores %repetidos en el vector.

ValorObjetivo = sum(1:nReinas-1);
fprintf('ValorObjetivo: %d\n',ValorObjetivo);

% Poblacion inicial
Poblacion=GeneraPoblacionInicial(NIND,nReinas,'GeneraPosicion');
ObjV = EvaluaPoblacion(Poblacion,'EvaluaPosicion');

% Para el gráfico
close;
grid;
hold on;
axis([1 MAXGEN ValorObjetivo/2 ValorObjetivo]);

% Inicializaciones
FinessMedio = zeros(MAXGEN,1);
MejorFitness = zeros(1,1);
mejorIndividuo = zeros(MAXGEN,nReinas);
gen = 0;
while ((gen < MAXGEN) & (max(mejorFitness) ~= ValorObjetivo)),

Poblacion=FormaNuevaPoblacion(Poblacion,ObjV,porcNewInd, funSeleccion, 'CruzaPosicion', 'MutaPosicion',pMutacion);
```

```
ObjV= EvaluaPoblacion(Poblacion,'EvaluaPosicion');

% Almacenar media del fitness y mejor individuo
mejor = find(ObjV==max(ObjV));
mejorFitness(gen+1) = ObjV(mejor(1));
mejorIndividuo(gen+1,:) = Poblacion(mejor(1),:);
fitnessMedio(gen+1) = mean(ObjV);

% Incrementar el numero de generaciones
gen = gen + 1;

% Actualizar gráfico
plot(mejorFitness,'-b');
plot(fitnessMedio,'-r');
if gen == 1
   legend('Mejor Fitness', 'Media de la Poblacion');
end
drawnow;
end
hold off
val = PintaTablero( mejorIndividuo(gen,:),'EvaluaPosicion' )
```

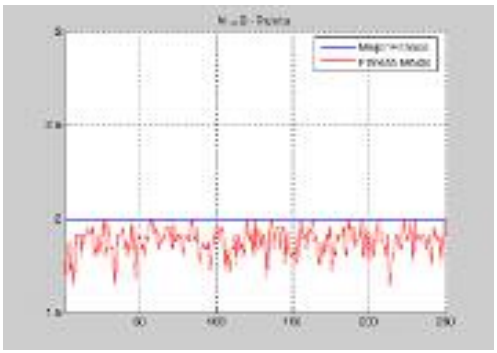
Los pasos que sigue esta implementación son los siguientes:

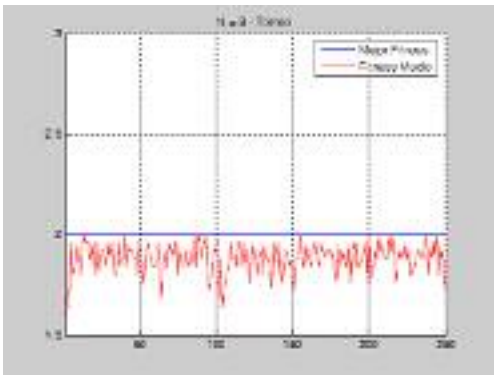
1. ***Se calcula el valor de fitness objetivo***.
2. ***Se crea la población inicial*:** se crea una matriz de tamaño NIND x nReinas, donde cada vector fila representa los cromosomas de las posibles soluciones del problema. Se genera de forma aleatoria. Cada valor del vector indica la fila en la que se encuentra la reina en esa columna. No se permiten valores repetidos en el vector.
3. ***Se evalúa dicha población***: calcula el fitness de los individuos de la población, es decir, de cada fila cuenta los posibles ataques entre las reinas.
4. ***Se genera una nueva población a partir de la anterior***: en primer lugar se calcula el número de nuevos individuos. El resto se completará con elitismo: algunos de los mejores individuos son copiados hacia la siguiente generación. Después se seleccionan los padres y se cruzan. Se han implementado dos operadores de selección: Ruleta y Torneo. Por último, se insertan uno o dos hijos, y se mutan los individuos creados (de acuerdo al porcentaje de mutación).
5. ***Se evalúa la nueva población***: se almacena la media del fitness y del mejor individuo, para poder representarlo de forma gráfica.
6. Se repiten los pasos 3 y 4 hasta que se alcanza el número máximo de generaciones o hasta que se alcanza el objetivo.

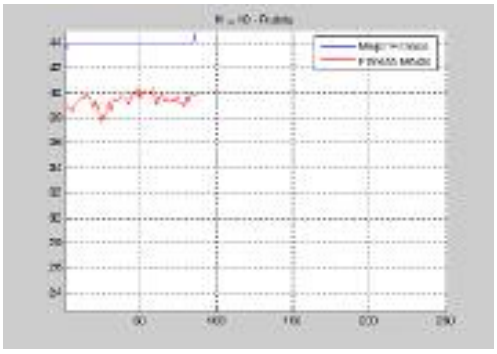
Comenzamos ejecutando el programa con los siguientes parámetros:

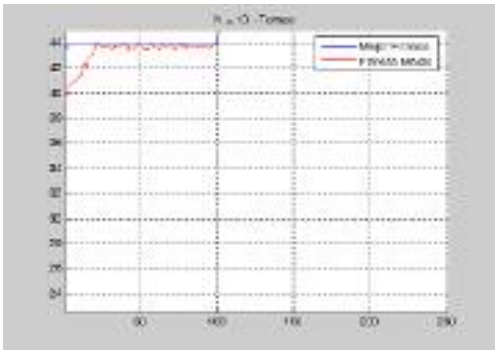
- Número de reinas: [3 10]
- Número de individuos de cada generación: 50
- Número máximo de generaciones: 250
- Porcentaje de nuevos individuos en la siguiente generación: 0.97
- Porcentaje de mutación: 0.1
- Función de selección: Ruleta, Torneo

Para un número de reinas entre 4 y 9, la convergencia es prácticamente instantánea: la generación en la que se alcanza el fitness buscado es muy pequeña. Para 10 reinas, la convergencia es mucho más lenta y se alcanza muchas generaciones después de la inicial. Sin embargo, para 3 reinas, no se llega a alcanzar el valor de fitness deseado.









Vemos que para 3 reinas, el fitness medio es muy variable para cada una de las generaciones y que en contadas ocasiones se alcanza el valor del mejor fitness. Para 10 reinas, esta situación se mantiene, aunque es más evidente la diferencia entre ambos valores si elegimos como función de selección la ruleta.

No obstante y como idea global, el algoritmo resuelve el problema encontrando, como es lógico, la solución antes ó después en función de la complejidad del problema.

11. REFERENCIAS

[1] Adam Marczyk. Algoritmos genéticos y computación evolutiva.

[2] Juan Julián Merelo Guervós. Informática evolutiva: Algoritmos genéticos

[3] Eduardo Morales Manzanares. Algoritmos Genéticos.

[4] Javier Campos . Algoritmos Genéticos.

[5] Mealnie Mitchell. An Introduction to Genetic Algorithms.

[6] Darrel Whitley. A Genetic Algorithm Tutorial.

[7] Bill Butler. The N-queens problem.