Distintas variantes de diseño de Algoritmos Genéticos y de Optimización de su funcionamiento

Julián Dorado

Servicios Informáticos de Apoyo a la Investigación (SIAIN) de la UDC.

Nieves Pedreira

Departamento de Computación de la UDC.

1.- Descripción

Los Algoritmos Genéticos (en adelante AG) se basan en el proceso de la evolución natural que tan buenos resultados le da a la propia Naturaleza, a la vista de la gran cantidad de organismos diferentes que sobreviven y la adaptación al medio que llegan a conseguir.

La mayoría de los organismos evolucionan mediante dos procesos primarios: selección natural y reproducción sexual. El primero determina qué miembros de una población sobreviven para reproducirse, y el segundo asegura la mezcla y recombinación de los genes de la descendencia. Cuando esperma y óvulo se unen, enfrentan las cadenas de sus cromosomas y las cruzan en algunas de sus partes, intercambiando, de esta forma, material genético. Esta mezcla permite que las criaturas evolucionen mucho más rápidamente de lo que lo harían si cada generación tuviera sólo la copia de los genes de un único padre, modificándola sólo ocasionalmente debido a una mutación.

La selección natural es un mecanismo muy simple: si un organismo no pasa algunas pruebas de adaptación, como reconocer a un predador y huir, muere. De manera similar, en informática y específicamente trabajando con algoritmos genéticos, aquellos algoritmos que sean poco eficientes se suprimirán en favor de los que mejor se adapten al problema que intentan resolver. Así, si se supone que un programa debe ordenar

números, por ejemplo en orden ascendente, simplemente se necesita comprobar si cada valor a la salida del programa es mayor que el anterior.

2.- Historia

Los primeros intentos para relacionar informática y evolución, a finales de los cincuenta y principios de los sesenta, tuvieron poco éxito porque, bansándose en los textos de biología del momento, creían en la mutación en lugar del cruce para producir nuevas combinaciones de genes. Fue entonces, a principios de los sesenta, cuando Hans J. Bremermann de la Universidad de California-Berkeley añadió una nueva forma de cruce. En ella, las características de la descendencia se determinaban sumando los genes correspondientes de los dos progenitores. Lamentablemente, este procedimiento de cruce estaba limitado, porque sólo se podía aplicar a características que se pudieran sumar de alguna manera significativa.

En esta época, John Holland había estado investigando en el análisis matemático de la adaptación y se había convencido de que la recombinación de grupos de genes mediante el cruce, era la parte más importante de la evolución. A mediados de los sesenta había desarrollado una técnica de programación, el algoritmo genético, que se adaptaba a la evolución tanto por cruce como por mutación. Durante la década siguiente trabajó para ampliar el alcance de los algoritmos genéticos creando un código genético que pudiera representar la estructura de cualquier programa informático.

El resultado fue el sistema clasificador, consistente en un conjunto de reglas, cada una de las cuales realiza acciones particulares cada vez que sus condiciones se satisfacen por alguna información. Las condiciones y acciones se representan por cadenas de bits que se corresponden con la presencia o ausencia de características específicas en las entradas y salidas de las reglas. Por cada característica presente, la cadena contendría un 1 en la posición correspondiente, y por cada una ausente, contendría un 0. El programador debería elegir las características más simples y primitivas que se puedan combinar para clasificar un amplio rango de objetos y situaciones. Aunque destacan en el reconocimiento, estas reglas se pueden hacer también para desencadenar acciones asociando bits a su salida. Cualquier programa que se pueda escribir en

un lenguaje de programación estándar como FORTRAN o LISP puede reescribirse como un sistema clasificador.

3.- Funcionalidad

Para producir reglas clasificadoras que resuelvan un problema particular, se empieza con una población de cadenas aleatorias de 1's y 0's y se valora cada cadena de acuerdo con la calidad de su resultado. Dependiendo del problema, la medida de ajuste podría ser la rentabilidad comercial, el momento decisivo de juego, porcentaje de error, etc. Después de valorarlas, las cadenas de alta calidad se unen y las de baja calidad perecen. Al ir pasando las generaciones, predominarán las cadenas asociadas a las mejores soluciones. El proceso de cruce combinará continuamente esas cadenas de diferentes formas, generando evoluciones cada vez más sofisticadas. Los tipos de problemas que han llevado a la práctica, van desde el desarrollo de nuevas estrategias en teoría de juego hasta el diseño de complejos sistemas mecánicos.

En el lenguaje característico de los algoritmos genéticos, la búsqueda de una buena solución para un problema es una búsqueda de cadenas binarias concretas. El universo de todas las cadenas posibles se puede considerar como un paisaje imaginario, donde los valles indican la localización de cadenas que codifican soluciones pobres, y los puntos más altos del paisaje se corresponden con las mejores cadenas posibles. Las regiones del espacio de soluciones se pueden definir también considerando cadenas que tengan 1's ó 0's en posiciones determinadas (una especie de equivalente binario de las coordenadas de un mapa). El conjunto de todas las cadenas que empiezan con un 1, por ejemplo, constituyen una región del conjunto de posibilidades. Lo mismo sucede con las cadenas que empiezan con un 0 ó que tienen un 1 en la cuarta posición, un 0 en la quinta y un uno en la sexta, etcétera.

Una técnica convencional para explorar este paisaje es el "hill-climbing", que consiste en empezar en un punto aleatorio, y si una pequeña modificación mejora la calidad de la solución, se continúa en esa dirección; en otro caso, se va en la dirección opuesta. Desgraciadamente, los problemas complejos originan paisajes con múltiples picos y, al aumentar el número de dimensiones del espacio del problema, pueden aparecer túneles, puentes e incluso figuras topológicas más complejas.

Encontrar la colina correcta o, incluso, determinar el camino adecuado se hace cada vez más difícil. Además, estos espacios de búsqueda suelen ser enormes. Por ejemplo, si cada movimiento de ajedrez tiene un alcance de 10 alternativas, y una partida típica dura 30 movimientos de cada jugador, habrá unas 10^{60} estrategias para jugar al ajedrez (la mayoría de ellas malas).

La forma de actuar de los algoritmos genéticos ante este tipo de problemas es situar una especie de red sobre este paisaje, donde cada nodo sería un punto valorado por el algoritmo. Además, la multitud de cadenas de una población la muestrea en varias regiones simultáneamente. También es muy importante que la proporción en la que el algoritmo genético muestrea regiones diferentes se corresponde directamente con la media de 'elevación' de las regiones (es decir, la probabilidad de encontrar una buena solución en esa zona). Esta habilidad de los algoritmos genéticos de enfocar su atención en las partes más prometedoras de un espacio de soluciones, es una consecuencia directa de su habilidad de combinar cadenas que contienen soluciones parciales.

El funcionamiento de los AG´s es muy simple y se basa en la repetición de tres pasos básicos en cada generación. En primer lugar, cada cadena de la población se evalúa para determinar el rendimiento de la estrategia que codifica. En segundo lugar, las cadenas de mayor categoría se cruzan. Para ello, se alinean dos cadenas, se elige una posición aleatoriamente, y las partes a la izquierda de esa posición se intercambian para producir dos nuevos descendientes. Uno contendrá los símbolos de la primera cadena hasta el punto de cruce y los de la segunda a partir de ahí, y el otro contendrá el cruce complementario. La descendencia no sustituye a las cadenas padres; en lugar de esto, reemplazan cadenas poco adecuadas, que son descartadas en cada generación para que la población total mantenga el mismo tamaño (refiriéndose esto a las versiones originales de AG desarrolladas por John Holland aunque, como se verá más adelante, tanto la forma de reemplazo, como el tamaño de las poblaciones pueden variar). En tercer lugar, las mutaciones modifican una pequeña parte de las cadenas, pasando de 0 a 1, o viceversa, algunos bits de unos pocos individuos de la población. La mutación, por sí sola, no suele avanzar en la búsqueda de una solución, pero sirve para prevenir el desarrollo de una población uniforme incapaz de evolucionar más.

El algoritmo genético explota las regiones más críticas, u 'objetivos' del espacio de soluciones, debido a que las sucesivas generaciones de reproducción y cruce dan lugar a un número cada vez mayor de cadenas en esas regiones. El algoritmo favorece a las cadenas más adaptadas como padres, y las cadenas que están por encima de la media (que caen en regiones objetivo) tendrán más descendencia en la generación siguiente. El número de cadenas en una región determinada aumenta de manera proporcional a la estimación estadística de la adaptación de esa región. Un estadístico necesitaría evaluar docenas de muestras de miles de millones de regiones para estimar la adaptación media de cada región, mientras que el algoritmo genético puede alcanzar el mismo resultado con muchas menos cadenas y casi sin cálculo.

3.1.- Definición de regiones

La clave de este comportamiento sorprendente es el hecho de que una única cadena pertenece a todas las regiones en las que aparecen algunos de sus bits. Por ejemplo, la cadena 11011001 es un miembro de las regiones 11****** (donde * indica un valor sin especificar), 1******1, **0**00* y así sucesivamente. Las regiones más grandes (las que contienen más bits sin especificar) serán muestreadas con una fracción grande de las cadenas de la población. Por tanto, un algoritmo genético que maneja una población de unos cuantos miles de cadenas, muestrea un número bastante mayor de regiones. Este paralelismo implícito le da al algoritmo genético su mayor ventaja frente a otros procesos resolutores de problemas.

El cruce complica los efectos del paralelismo implícito. El propósito del cruce de cadenas en el algoritmo genético es examinar partes diferentes de regiones objetivo, en lugar de examinar constantemente la misma cadena en generaciones sucesivas. Pero el proceso también puede trasladar una descendencia de una región a otra, modificando el porcentaje con que se muestrean diferentes regiones, alejándose de la proporcionalidad estricta para calcular la media de adaptación. Esta desviación ralentizará la tasa de evolución.

La probabilidad de que la descendencia de dos cadenas salga de la región de sus padres depende de la distancia entre los 1's y los 0's que definen la región. La descendencia de una cadena 10****, por ejemplo,

puede estar fuera de esa región sólo si el cruce empieza en la segunda posición de la cadena (una posibilidad entre cinco para una cadena de seis genes). La descendencia de una cadena de seis genes contenida en la región 1****1 dejará la región de sus padres sin importar en qué posición se produzca el cruce (dependiendo sólo de que bits contenga en esas posiciones el individuo con el que se cruce).

Los 1's ó 0's adyacentes que definen una región se denominan bloques compactos o "genes". Estos genes son los que, más probablemente, sobrevivirán intactos a un cruce y, por tanto, de propagarse en generaciones futuras con una tasa proporcional a la media de adaptación de las cadenas que los contienen. Aunque un mecanismo de reproducción que incluya el cruce no muestrea todas las regiones de manera proporcional a su adaptación, sí lo hace para todas las regiones definidas por bloques compactos. El número de bloques compactos de una población de cadenas excedería con mucho el número de cadenas y, así, el algoritmo genético muestra su paralelismo implícito.

Curiosamente, existe una operación en genética natural llamada "inversión", que ocasionalmente reordena genes, de manera que aquellos que estaban separados en los padres pueden estar juntos en la descendencia. Esto viene a ser como redefinir un bloque para que sea más compacto y menos sujeto a una ruptura por cruce. Si el bloque especifica una región con una media alta de adaptación, la versión más compacta desplaza automáticamente a la menos compacta ya que pierde menos descendencia a causa del error. Como resultado, un sistema adaptativo que utilice la inversión puede descubrir y favorecer versiones compactas de bloques útiles.

4.- Problemas no lineales

El paralelismo implícito de los algoritmos genéticos permite examinar y explotar un gran número de regiones en el espacio de búsqueda, manipulando relativamente pocas cadenas. El paralelismo implícito también ayuda a los AG a hacer frente a problemas "no lineales" (en los que la adaptación de una cadena que tenga dos bloques concretos pueda ser mucho mayor, o mucho menor, que la suma de las adaptaciones atribuible a cada bloque por separado.

Los problemas lineales presentan un espacio de búsqueda reducido, porque la presencia de un 1 o un 0 en una posición de una cadena no repercute en la adaptación que implica la presencia de un 1 o un 0 en cualquier otra posición. El espacio de cadenas de 1.000 dígitos, por ejemplo, tiene más de 31.000 posibilidades; pero, si el problema es lineal, un algoritmo necesita examinar sólo las cadenas que tienen un 1 o un 0 en cada posición, lo que hace un total exacto de 2.000 posibilidades.

Cuando un problema es no lineal, la dificultad se incrementa. La media de ajuste de cadenas de la región *01***, por ejemplo, podría estar sobre la media de la población, a pesar de que las adaptaciones asociadas a *0**** y **1*** están por debajo de la media. La no linealidad no significa que existan bloques inútiles, sino, simplemente, que los bloques que consisten en únicos 1's ó 0's no son adecuados. Esa característica conduce a una explosión de posibilidades: el conjunto de todas las cadenas de 20 bits de longitud tiene más de 3.000 millones de bloques. Afortunadamente, todavía se puede explotar el paralelismo. En una población de unos pocos miles de cadenas, aparecerán bloques compactos en 100 cadenas o más, lo suficiente para obtener una buena muestra estadística. Los bloques que explotan la no linealidad para obtener un rendimiento por encima de la media se utilizarán automáticamente más a menudo en generaciones futuras.

5.- Explotación VS exploración

Además de todo lo que ya se ha visto, los AG ayudan a solucionar un problema que resultaba muy complicado para otros métodos de resolución de problemas, como es alcanzar un equilibrio entre la exploración y la explotación. Para ilustrar este problema podemos fijarnos en el juego del ajedrez. Dentro de una partida, una vez que se encuentra una buena estrategia, es posible concentrarse en explotar esa estrategia. Pero esta elección acarrea un coste adicional debido a que la explotación hace improbable el descubrimiento de estrategias nuevas. De intentar cosas nuevas, aunque probablemente sean complicadas, se suelen obtener mejoras. Como la mayoría de estos intentos pueden fallar, la exploración implica una degradación del rendimiento. El decidir hasta qué grado se debe arriesgar el presente por el futuro es un problema clásico para todos los sistemas que se adaptan y aprenden.

La aproximación del algoritmo genético a este obstáculo se traduce en el "cruce". Aunque el cruce puede interferir con la explotación de un bloque al romperlo, este proceso de recombinación prueba estos bloques en nuevas combinaciones y nuevos contextos. El cruce genera nuevos ejemplos de regiones que están por encima de la media, confirmando o desaprobando las estimaciones producidas por ejemplos anteriores. Cuando el cruce rompe un bloque, se produce un bloque nuevo que permite al algoritmo genético examinar regiones que no se habían muestreado con anterioridad.

6.- Distintas formas de aumentar el rendimiento de los Algoritmos Genéticos

Como se ha visto, los AG pueden ser muy eficientes encontrando soluciones óptimas en una gran variedad de problemas. Esta innovadora técnica funciona especialmente bien resolviendo problemas complejos del mundo real ya que no impone la mayoría de las limitaciones de las técnicas tradicionales. Sin embargo, las mismas características que hacen que los AG sean tan robustos los convierten en algoritmos de computación intensiva y, por tanto, más lentos que los otros métodos. Aquí se presentan cinco estrategias [2] independientes del problema que pueden mejorar la eficiencia de cualquier AG de propósito general.

6.1.- Reemplazar a Darwin con Lamarck

Los AG 'clásicos' (desarrollados por John Holland) se inspiraron en la teoría de la selección natural de Darwin. Los AG generan muchas posibles soluciones (organismos) para un problema dado y, entonces, permiten que estos organismos compitan y se crucen para crear una descendencia incrementalmente mejor. En este sentido, los AG evolucionan el problema guiándolo hacia la solución óptima, en vez de intentar resolverlo directamente.

En el siglo XIX, la teoría de Darwin fue desafiada por Jean Lamarck, quien propuso que los cambios medioambientales a lo largo de toda la vida de los organismos causan cambios estructurales que son transmitidos a la descendencia. Esta teoría permite a los organismos transmitir el conocimiento y la experiencia que adquirieron a lo largo de su vida.

Mientras que ningún biólogo actual cree que los rasgos adquiridos en el mundo real puedan ser heredados, el poder de la teoría se ilustra con la evolución de las sociedades. Las ideas y los conocimientos pasan de generación en generación a través de un lenguaje estructurado y de una cultura. Debido a esto, la evolución de la sociedad va más allá que la de los individuos.

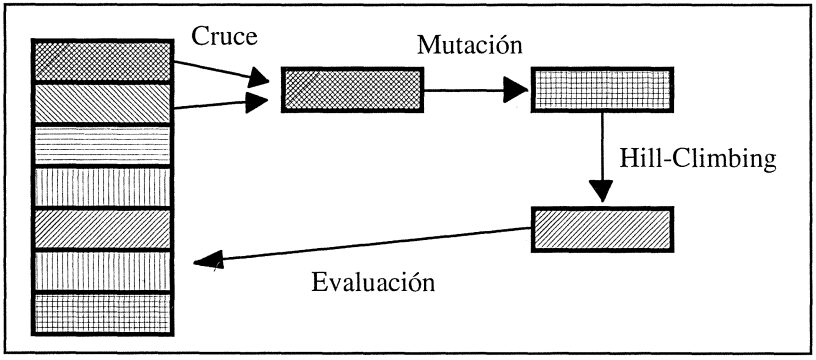


Figura 1.- Combinación de AG y Hill-Climbing

Nuestros organismos digitales pueden beneficiarse de las ventajas de la teoría de Lamarck, permitiendo que algunas de sus experiencias sean transferidas a futuros organismos. Así, se pueden mejorar las habilidades de los AG´s para enfocarlas en las áreas más prometedoras. Los AG´s convencionales emplean cruces y mutaciones para producir una descendencia de dos padres e inmediatamente pasar los organismos resultantes por la función de evaluación para determinar su grado de aptitud para la tarea. Siguiendo una aproximación más "Lamarquiana", primero intentaríamos inyectar algunas habilidades en la descendencia antes de que ésta sea evaluada. Se podría usar una rutina tradicional de "Hill-Climbing", como punto de comienzo para conseguir una veloz y localizada optimización (Figura 1).

Existen estudios que han demostrado que tal técnica *"AG/Hill-Climbing"* frecuentemente supera a cualquiera de estos métodos trabajando independientemente. Ya que, mientras un AG puede ser lento realizando el ajuste fino final, este método determina más rápidamente si el AG está enfocado en los caminos más prometedores. Si la función de

evaluación en ciertos problemas es demasiado ruidosa o discontinua para usar la técnica de Hill-Climbing convencional, puede usarse un AG local para afinar el ajuste de los organismos. Se puede crear una muy pequeña población de organismos similares alrededor de una descendencia y con mutaciones localizadas y operadores de cruce para favorecer pequeños cambios sobre los grandes, estos AG´s locales se comportarán como algoritmos de Hill-Climbing.

6.2.- Desarrollo de especies

Los AG tradicionales fuerzan a todos los organismos a evolucionar en una gran población. Dividiendo de forma arbitraria la población en varias minipoblaciones no se incrementará el rendimiento y puede incluso dificultarse la habilidad de los AG´s para converger (al disminuir la variabilidad genética). Este fenómeno es, generalmente, cierto ya que hay una gran probabilidad de que cada población evolucionará hacia soluciones similares, provocando una cantidad innecesaria de computación.

Una forma más efectiva de acelerar el rendimiento de un AG es construir un algoritmo genético de generación de especies ("species-generating genetic algorithm" SGGA). Las especies y subpoblaciones son creadas automáticamente cuando el AG detecta que dos áreas significativamente distintas están siendo exploradas dentro de una población. Para determinar cuando la población está lo bastante diversificada para que merezca la pena dividirla, se emplea el "análisis de clusters" o se generan nuevas especies cuando la distancia relativa entre los organismos mejor adaptados traspasa un cierto umbral. Cada organismo puede ser asignado a la especie con la cual esté más relacionado. Estas especies generalmente evolucionan indepen-dientemente unas de las otras aunque, ocasionalmente, se pueden intercambiar unos pocos de sus organismos (Figura 2).

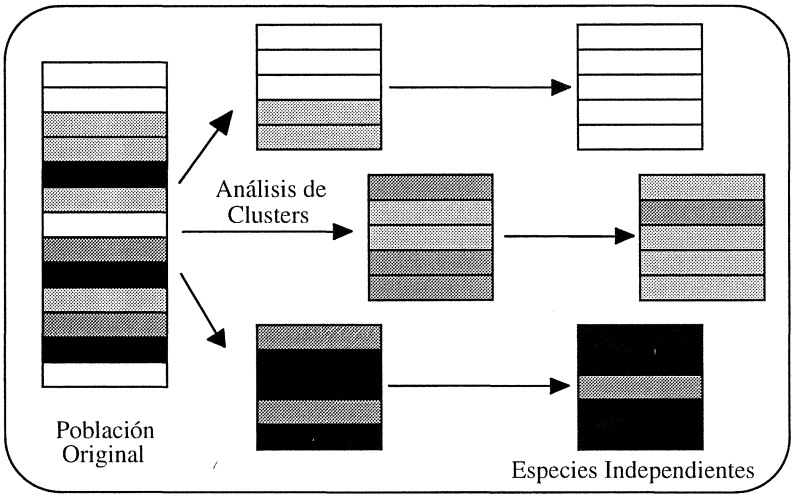


Figura 2.- Desarrollo de especies independientes

Un entorno donde emergen distintas especies de soluciones ofrece ventajas sobre el modelo tradicional de población única. En primer lugar, el algoritmo de población única pierde tiempo criando organismos de similares características, sin considerar las distancias relativas entre ellos. El SGGA permite que áreas locales significativamente diferentes sean exploradas en todo su potencial. El SGGA funciona especialmente bien cuando se incrementa el número de dimensiones en la función objetivo.

El SGGA también permite que cada especie independiente converja más rápidamente y pueda, por tanto, dar una temprana indicación de las áreas prometedoras. Las especies que funcionen peor desaparecerán, o serán remezcladas con especies similares para proporcionar tiempo de computación para las especies más prometedoras. Adicionalmente, en cada especie se pueden personalizar los parámetros de cruce y mutación para que funcione más eficientemente.

Finalmente, los SGGA están mejor equipados para funcionar en un entorno multiprocesador, tal como una red de área local. Debido a la naturaleza independiente de cada especie, cada máquina puede ser añadida o eliminada cuando sea necesario, sin que haga falta sincronizar las aplicaciones de procesamiento paralelo. Para empezar,

probablemente se necesitará una aplicación esclava para cada máquina y una aplicación controladora. Entonces se creará un sistema de mensajes que permita el intercambio de datos y mensajes. Este puede comenzar y parar la optimización aceptando un nuevo organismo, pidiendo un organismo que ya existía y devolviendo el mejor organismo que cada especie haya encontrado.

6.3.- Suavizado y escalado

La mayoría de los problemas del mundo real incluyen restricciones. Por ejemplo, cuando se intenta determinar el mejor diseño para la estructura de un puente, se podría estar limitado por el requerimiento de que la estructura debería soportar un mínimo de carga de 10 toneladas. Una vez que esta restricción se satisface, el problema a resolver es encontrar el diseño que use menos material.

Parece lógico pensar que el AG debería ser prevenido para que no considere soluciones no válidas. Sin embargo, este tipo de restricciones fuertes que son útiles en las técnicas de optimización tradicionales, no lo son tanto con los AG que basan su funcionamiento en la variedad genética de sus elementos. Si se eliminan organismos sólo porque no cumplen alguna restricción, entonces, se está eliminando información vital que, eventualmente, podría conducir a la solución óptima. Otra razón para evitar las restricciones fuertes es que, permitir pequeñas violaciones de las restricciones, puede producir grandes beneficios en otras áreas. Por ejemplo, el diseño de la estructura óptima puede soportar 9.9 toneladas pero, en contrapartida, usar sólo la mitad de materiales que el siguiente mejor diseño.

Por estas razones, generalmente, se acepta que reemplazar las restricciones fuertes por restricciones débiles bien seleccionadas incrementa el rendimiento de los AG´s. Las restricciones débiles no previenen que los AG exploren soluciones no válidas sino que penalizan a los organismos para desanimar los esfuerzos en esa dirección.

Cada restricción débil se representa en una parte de código llamada "función de penalización". Si se quiere penalizar el diseño de puentes que no soporten cargas de menos de 10 toneladas, una buena función de penalización podría reducir el ajuste calculado para el diseño en una cantidad proporcional a la cantidad de peso por debajo de las 10

toneladas que el puente es capaz de soportar. Por ejemplo, para un diseño que soporte 9.9 toneladas se reduciría su ajuste en 0.1 y a un diseño de 9.5 toneladas en 0.5.

La función de penalización también puede ser de tipo logarítmico, para forzar más rápidamente la convergencia al espacio de soluciones-restricciones. La cantidad de penalización debe tener un escalado relativo a los valores de ajuste de la función de evaluación. Si una población de organismos muestra unos valores de ajuste entre 3´4 y 3´6, las diferencias en el ajuste quedarán completamente difuminadas por una función de penalización que añada penalizaciones de 10.000 para diseños que casi cumplan una restricción.

6.4.- Añadir "backtracking" (vuelta atrás)

En algunos problemas, ciertas restricciones tienen que cumplirse obligatoriamente y, entonces, el espacio de búsqueda se vuelve tan limitado que, incluso un AG, debido a las fuertes penalizaciones, perdería la mayor parte del tiempo de computación explorando soluciones no válidas. El "backtracking" ofrece una forma sencilla de asegurar que se cumplen todas las restricciones rápidamente mientras que se mantiene la diversidad necesaria para que funcione el AG.

Un método tradicional, de restricciones fuertes, evaluará primero cada nuevo elemento de la descendencia, chequeando en él una lista de restricciones antes de incorporarlo a la población. Si el organismo ha violado alguna restricción, el proceso crea una nueva descendencia hasta que encuentre una solución aceptable. En un problema muy restringido, la mayor parte de la computación puede ser desperdiciada en encontrar un único organismo válido. Incluso, cuando se encuentran una pareja de organismos válidos, es muy probable que sus combinaciones generen muchos organismos no válidos.

El método de backtracking chequea, para cada descendiente, una lista de restricciones fuertes. Si la solución descendiente viola alguna restricción, el organismo hace un "backtrack" o "vuelta atrás", involucionando hacia el material genético de sus padres hasta que vuelve a cumplir todas las restricciones. Funcionalmente, el "backtracking genético" empareja continuamente a los padres originales con ratios incrementalmente más bajos de cruce y mutación. Cada vez que el

organismo realiza una vuelta atrás, se convierte en más parecido a uno de sus padres. Ya que ambos padres cumplen las restricciones, el organismo realizará la vuelta atrás hacia un espacio de soluciones-restricciones válido.

El "lazo de backtrack" es un procedimiento relativamente rápido. Diferentes investigaciones sobre este tema indican que el "backtracking" es un procedimiento más rápido que usar solamente una función de penalización, cuando la región de soluciones aceptables es pequeña o muy fraccionada. Además, el "backtracking" es el mejor método si las restricciones tienen que ser estrictamente cumplidas.

6.5.- Ajuste de Parámetros

La mayoría de las implementaciones de AG tienen sólo unos pocos parámetros configurables: tamaño de la población, ratio de cruce y ratio de mutación. Mientras que, utilizando valores fruto de la experiencia para estos parámetros, pueden obtenerse buenos resultados para un gran número de problemas, se puede conseguir doblar la velocidad ajustando estos parámetros a los valores correctos para cada problema concreto. La dificultad estriba en que estos valores correctos pueden cambiar drásticamente con sólo unas pocas modificaciones en el modelo de problema. Incluso cuando se optimiza el mismo modelo, los parámetros ideales cambiarán según avanza el proceso de optimización.

Por ejemplo, el cruce es frecuentemente más importante en las primeras generaciones de un AG, mientras que la mutación es más importante en las últimas etapas de ajuste (Figura 3). El tamaño de la población también debe cambiarse a lo largo del proceso para equilibrar la velocidad de la evolución con la diversidad genética. Los AG adaptativos aumentan el tamaño de la población (e incrementan la diversidad genética) cuando el progreso es lento y, a la inversa, disminuyen el tamaño de las poblaciones heterogéneas.

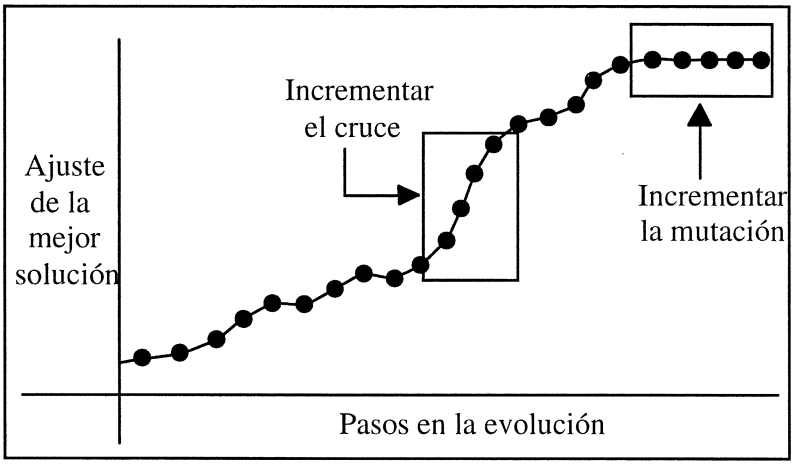


Figura 3.- Curva de ajuste cruce-mutación.

7.- Ejemplo de aplicación de los AG al entrenamiento de RNA

En este punto, se va a presentar el funcionamiento y la construcción de la aplicación AG-RNA desarrollada en el Laboratorio de Redes de Neuronas Artificiales y Sistemas Adaptativos de la Facultad de Informática de la Universidade da Coruña, demostrando las posibilidades de una novedosa tecnología (los AA.GG.) para el entrenamiento genérico de RNA. Ha sido desarrollado para usarse en una plataforma PC utilizando Windows 3.1 o posterior. Para su implementación, se ha utilizado, por un lado, Visual Basic 4.0 en el desarrollo de la interface: y, por otro, Visual C++ 1.51 en el desarrollo de la librería que implementa el AG. Dentro de esta librería, la función de evaluación que se utiliza estima la adaptación de una RNA (de alimentación hacia delante) utilizando los pesos proporcionados por el AG y los ficheros de entrenamiento desarrollados con esta misma herramienta.

La librería con las funciones del AG se ha compilado para obtener un fichero en formato "dll" (dynamic link library) accesible tanto desde programas en Visual Basic como el aquí citado, como desde otros desarrollados en lenguaje C o Pascal.

El entorno del programa (Figura 4) permitirá modificar con facilidad los parámetros de la configuración, tanto del AG como de la RNA. Además, en la pantalla principal se muestra la evolución del aprendizaje con un diagrama de las conexiones de la red y una evaluación de los mejores individuos de la población.

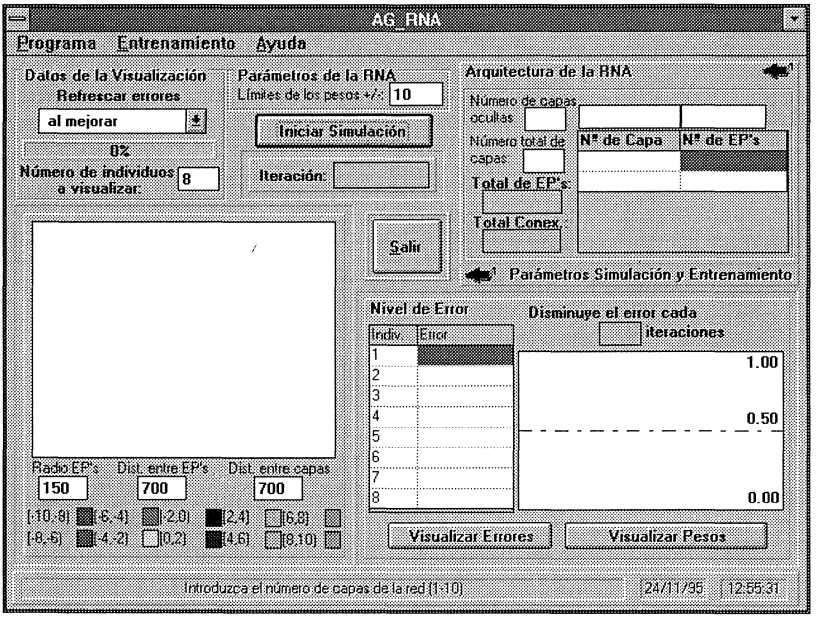


Figura 4.- Pantalla principal del programa AG-RNA

Vamos a ver, ahora, cuales son los pasos que se tienen que dar para conseguir entrenar una RNA utilizando este programa, desde conseguir un fichero de entrenamiento hasta la interpretación de los resultados finales. El primer paso que hay que dar para entrenar una red con este programa es construir un fichero de datos con el que el AG pueda evaluar la adaptación de la RNA a ese conjunto de entrenamiento. Para entrar en la pantalla diseñada para realizar esta labor (Figura 5) hay que pulsar la opción "Construir Fichero" en el menú "Entrenamiento". Una vez dentro de esta pantalla se encuentran las opciones habituales para manejar ficheros: *Nuevo, Abrir, Cancelar* o *Salvar*. Para la construcción del nuevo

fichero primero hay que cubrir los campos de número de columnas, filas y ejemplos y pulsar después el botón *Nuevo*. Después de hacer esto aparece una tabla vacía con el número de columnas, filas y ejemplos solicitados. Para rellenar estos campos basta con pulsar primero en la casilla que se quiere rellenar y escribir después su valor en el campo de texto que se sitúa encima de la tabla.

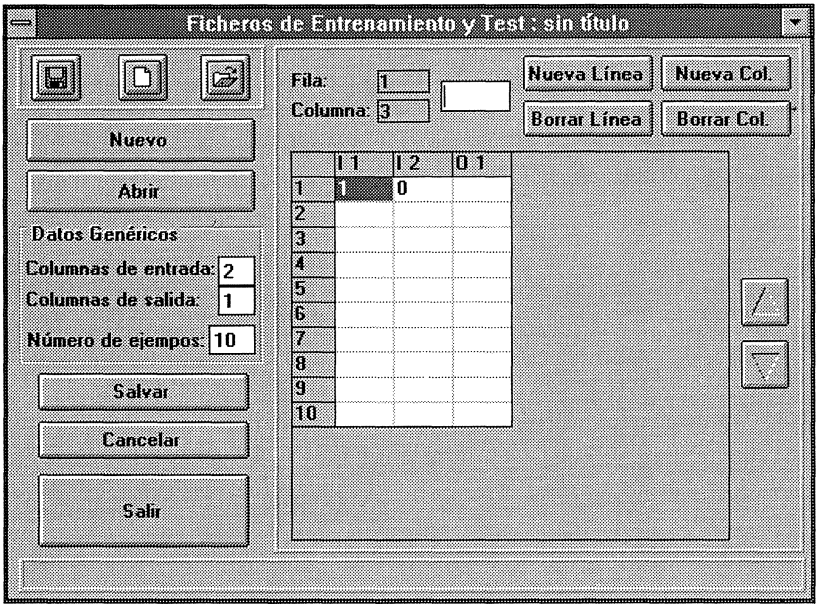
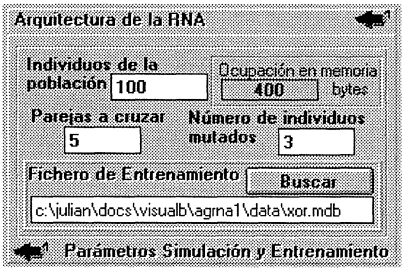


Figura 5.- Pantalla para crear y modificar ficheros de entrenamiento.

Como se puede ver, en la Figura 5, también es posible aumentar o disminuir columnas y filas una vez que hemos comenzado la edición de la tabla de valores, pulsando los botones correspondientes de la esquina superior derecha de la pantalla. Cuando tengamos todos los datos que queremos incluir en el fichero de entrenamiento, pulsamos el botón Salvar para que construya una Base de Datos en formato Access 1.0. Este formato es de fácil lectura y modificación desde un gran número de aplicaciones comerciales, incluso es posible editar el fichero base que se

adjunta (simple.mdb) con cualquier programa de Base de Datos que soporte este formato, y construir directamente el fichero de entrenamiento. Una vez cumplido el trámite de construir el fichero podemos pasar a configurar los parámetros del AG, de la RNA y del entorno de visualización.

Los parámetros del AG que se pueden configurar son los ya comentados en la primera parte del capítulo (enunciados en el primer estándar de Holland). Así, se puede especificar el tamaño de la población, el ratio de cruce y el de mutación. En esta pantalla es, también, donde se le dice al programa cual es el fichero que se debe utilizar para evaluar el nivel de entrenamiento de la red. Con respecto a los parámetros de la RNA, tenemos el número de capas y el número de neuronas por capa. (Figuras 6 y 7).



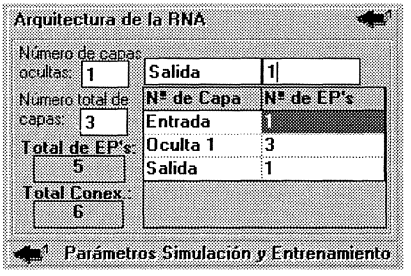


Figura 6.- Parámetros del AG. Figura 7.- Parámetros de la RNA

Una vez configurados estos parámetros, únicamente nos quedan por ajustar los valores referidos a la visualización y a la RNA que se pueden considerar opcionales. En la Figura 8 se muestran los controles donde podemos modificar estos parámetros. En primer lugar, para acelerar la simulación, podemos Refrescar los valores de salida y de error cada generación, cada 10, cada 100, o cuando se produzca un individuo nuevo que supere al mejor de los que ya existían. Obviamente, el proceso de simular es mucho más rápido si evitamos repintar toda la información gráfica de las redes en cada generación. Además, no perdemos información si evitamos visualizar aquellos pasos en que no se haya producido mejoría. En el caso de que se opte por una opción fija (10 ó 100), también se visualiza que tanto por ciento de esas generaciones ya se

han desarrollado. Para los controles de visualización de resultados y error que se verán a continuación, se especifica, aquí, el número de individuos a visualizar (también para configurar el tiempo de respuesta), limitando el número de individuos de los que se visualizarán datos. El control *Parámetros de la RNA* se refiere al límite superior e inferior que pueden alcanzar los pesos de las conexiones.

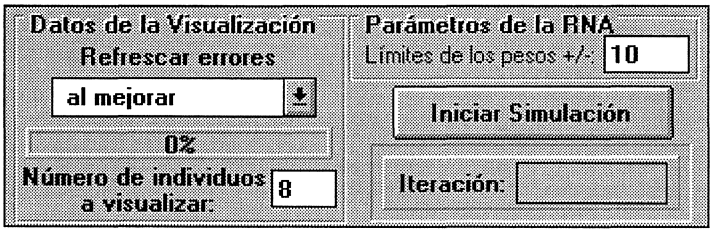


Figura 8.- Parámetros genéricos de visualización y RNA.

Por último, en el campo Iteración se muestra el número de generaciones que ya se han desarrollado durante la simulación. Para comenzar la simulación no queda más que pulsar el botón Iniciar Simulación (Figura 8), con lo que se empezarán a visualizar los resultados de los individuos generados.

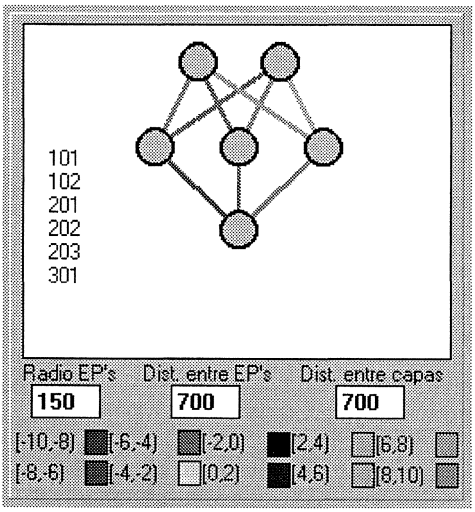


Figura 9.- Gráfico de la mejor red generada hasta ese momento.

Una vez que comienza el proceso de simulación, se obtienen, de una forma directa, dos tipos de informaciones sobre la evolución de los individuos con el paso de las generaciones. En un gráfico de la red (Figura 9), se va a reflejar una aproximación de los pesos de las conexiones mediante un código de colores para obtener una primera impresión de como evoluciona la simulación. La RNA que refleja el gráfico es siempre la que, dentro de la población que desarrollamos, tiene un nivel más alto de adaptación al fichero de entrenamiento.

La RNA se supone que está representada con la capa de entrada en la parte superior de la pantalla y la de salida en la inferior. El gráfico es configurable tanto en el tamaño y posición de los Elementos de Proceso (EP) o neuronas de la RNA, cambiando los valores "por defecto" que se observan en la figura, como en los colores de las conexiones, pulsando dos veces en los recuadros coloreados de la parte inferior.

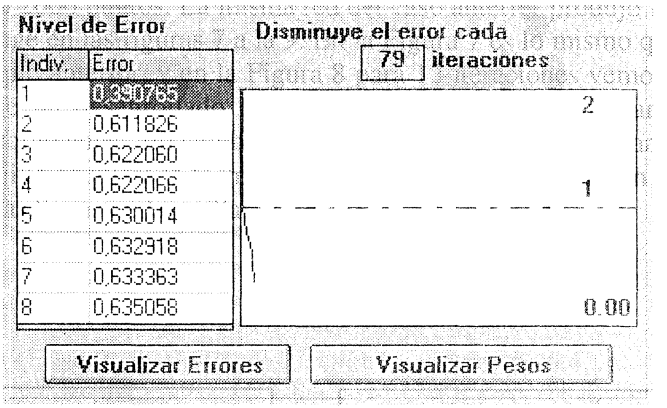


Figura 10.- Visualización de los datos sobre el error cometido

La Figura 10 nos muestra, ya, datos exactos sobre los individuos a lo largo de la simulación. En la parte izquierda, se ven los errores acumulados entre la salida obtenida y la salida deseada para cada uno de los casos del fichero de entrenamiento para los "n" mejores individuos de la población (en este caso 8). En la parte derecha, se dibuja también, en tiempo real, una gráfica de la disminución del error del mejor individuo de la población, representando la evolución del proceso de aprendizaje de

la red. Encima de esta gráfica, se nos da información sobre el número de generaciones que han transcurrido desde que se mejoró al mejor individuo que existía hasta el momento. Este valor es muy importante para ponderar la posibilidad de saturación del proceso de aprendizaje, que podría hacerla caer en un estado de pérdida de generalización.

Los botones de la parte inferior de la Figura 10 permiten acceder a información más detallada. En Visualizar Errores, se ofrece para cada individuo de la población, la salida deseada, la obtenida y el error que se genera para cada caso del conjunto de entrenamiento. También, se calcula el error total que es el que se visualiza en la pantalla de la Figura 10. En "Visualizar Pesos", para cada individuo, se pueden observar los pesos de las conexiones entre los EP, tal como se pueden ver en el gráfico de la RNA, pero en un formato numérico.

7.1.- Resultados

Con el desarrollo de este programa, se ha querido explorar o realizar una primera aproximación al desarrollo de sistemas híbridos que combinen técnicas, tan dispares en su concepción, como las Redes de Neuronas Artificiales y los Algoritmos Genéticos. Estas técnicas tienen, sin embargo, dos puntos importantes en común. En primer lugar, ambas se desarrollan como técnicas que emulan comportamientos de la naturaleza y, en segundo lugar, las dos funcionan como algoritmos de búsqueda en espacios de soluciones grandes y complejos donde, además, suele ser precisa una cierta capacidad de adaptación y de trabajar con datos: incompletos, imprecisos, inconsistentes e inciertos.

Los resultados obtenidos con esta experiencia han demostrado la gran utilidad que pueden llegar a tener los AG en el entrenamiento de RNA's de tipo convencional (alimentación hacia adelante en el caso que se ha visto), así como avanzar las ventajas que se pueden obtener de su futura aplicación a RNA de arquitecturas más complejas. Esta ventajas se deben, sobre todo, a que es posible obviar el desarrollo de algoritmos de entrenamiento distintos que posibiliten trabajar con cada arquitectura de red que se desarrolle. Si utilizamos los AG para realizar este tipo de tareas, variando únicamente la función de evaluación para cada tipo de red, podremos conseguir redes entrenadas y, por tanto, evaluaciones de las nuevas arquitecturas en muy cortos períodos de tiempo.

NOTA

Si tiene interés por comprobar las posibilidades del programa AG-RNA puede conseguirlo a través del servicio de FTP anónimo de la Univesidade da Coruña, en la máquina ftp.udc.es dentro del directorio /pub/department/computacion/rnasa-lab/ag-rna.zip. Para cualquier consulta sobre su instalación o funcionamiento pueden contactar con nosotros en la direcciones de correo electrónico: cijulian@udc.es y nieves@udc.es

Bibliografía

[1] Holland J. H., *Genetic Algorithms*, Scientific American, Julio 1992, 66-72.

[2] Kennedy S. A., *Five Ways to a Smarter Genetic Algorithms*, AI Expert, Diciembre 1993, 35-38.

Lecturas recomendadas

- L. Davis, *Handbook of Genetic Algorithms*, Van Nostrand Reinhold (New York) 1991.
- L. Davis, *Genetic Algorithm Profiles in Advanced Tecnology for Developers*, High-Tech Communications (Sewickeley) 1992.
- D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, Adisson-Wesley, 1989.
- J. H. Holland, *Adaptation in Natural and Artificial Systems*, MIT Press, 1992.

  Genetic Algorithms: Proceedings of the Forth International Conference, Morgan Kaufmann.