Computación neuronal. Universidade de Santiago de Compostela, 1995 (385-424)

XIII

DISEÑO DE REDES NEURONALES ARTIFICIALES MEDIANTE ALGORITMOS GENÉTICOS

Francisco J. Marín; Francisco Sandoval
Universidad de Málaga

1. INTRODUCCIÓN

Uno de los problemas a los que más investigación se está dedicando últimamente es a encontrar la topología idónea de una red neuronal (organización, número de neuronas y conectividad entre las neuronas) que resuelva una determinada tarea. Desde que se descubrió que el perceptron [Minsky y Papert, 1988] no podía resolver problemas que no eran linealmente separables y que para ello era necesario colocar capas ocultas entre la capa de entrada y la de salida, el esfuerzo se centró en saber cuantas capas ocultas y cuantas neuronas en cada capa eran necesarias para resolver un determinado problema. Hasta ahora, la decisión de elegir la topología de la red se realiza mediante la tediosa tarea de prueba y error, la cual hace que el tiempo necesario para alcanzar una aceptable solución sea excesivamente largo, empleando un gran tiempo de CPU. Desgraciadamente, no existe una metodología general que resuelva el problema, y el diseñador se basa en su propia experiencia para fijar la topología de la red neuronal.

Otro gran problema en el diseño es la falta de escalado que presentan las redes neuronales; la elección de una buena topología de la red para un determinado problema puede no ser buena si se aumenta la complejidad del problema. Por ejemplo, si se elige la topología de la red que resuelva el problema de la paridad de tres bits, no se puede garantizar que para N-bits, si N es un número grande, siga siendo la correcta sin más que aumentar el número de neuronas ocultas. La mayoría de las investigaciones se testean con pequeños problemas, llamados problemas-

juguete, no siendo extrapolables para problemas de mayor envergadura. Por otra parte, en una red neuronal existe gran cantidad de información redundante contenida en los pesos; una red óptima debe eliminar dichos pesos redundantes manteniendo la capacidad funcional de la red. Este proceso se conoce como podaje de la red [Karnin, 1990].

Existen bastantes algoritmos que intentan reducir el número de unidades ocultas. Estos algoritmos del tipo constructivos/destructivos [Fahlman y Lebiere, 1990; Frean, 1990; Le Cun y col., 1990 y Hirose y col., 1991] empiezan con una posible topología de la red y mediante pequeños ajustes se añaden o eliminan neuronas de las capas ocultas durante el proceso de aprendizaje. Estos métodos dependen fuertemente de la topología inicial de la red y presentan ciertas facilidades para caer en mínimos locales, no garantizando una solución satisfactoria o presentando convergencias demasiado lentas o demasiado rápidas. Además, como la estructura de la red está fuertemente ligada a la existencia de un más o menos potente algoritmo de aprendizaje, no se puede garantizar que el proceso de aprendizaje de la red sea el más idóneo para un determinado algoritmo de aprendizaje y, por otra parte, si el proceso de aprendizaje no da buenos resultados, no es posible saber si ha fallado la red o el algoritmo de aprendizaje. La relación entre la topología de la red y el proceso de aprendizaje es generalmente desconocido; existen estudios a nivel microscópico en el espacio de pesos que intentan encontrar la estrategia de aprendizaje más idónea para una determinada red [Joya y col., 1993].

Los algoritmos genéticos son algoritmos evolutivos de búsqueda inspirados en procesos de selección natural [Holland, 1975; Goldberg, 1989]. Se aplican principalmente en problemas de optimización y como mecanismos para describir reglas en aplicaciones de aprendizaje de máquinas. Se comportan como una eficaz herramienta para tratar problemas que presenten una superficie compleja y ruidosa con múltiples mínimos locales y grandes espacios de búsqueda. Estas son las características del problema mencionado anteriormente, por lo que parece lógico aplicar algoritmos genéticos al diseño de redes neuronales.

La aplicación de los algoritmos genéticos al diseño de redes neuronales para resolver una determinada tarea está siendo dirigido desde tres campos de aplicación: primero, en la búsqueda del conjunto óptimo de pesos de conexión para una topología y regla de aprendizaje fijada de antemano [Whitley y Starkweather, 1990; Montana y Davis, 1989]; segundo, en la búsqueda de la topología óptima una vez fijada la regla de aprendizaje [Harp y col., 1989; Whitley y col., 1990; Radcliffe, 1993; Robbins y col., 1993]; y tercero en la búsqueda de la regla de aprendizaje óptima una vez que el número de neuronas y la conectividad entre ellas ha sido seleccionada [Fontanari y Meir, 1991].

2. PRINCIPIOS DE LOS ALGORITMOS GENÉTICOS

La figura 1 muestra la taxonomía de los métodos de búsqueda. Las técnicas de búsqueda se pueden clasificar en tres grandes grupos: Basadas en cálculos, enumerativas y aleatorias [Goldberg, 1989]. Las técnicas basadas en cálculos utilizan un conjunto de condiciones necesarias y suficientes que debe satisfacer la solución óptima. Se subdividen en directas, cuando se guía la dirección de búsqueda por el gradiente del nuevo punto (Hill-climbing), y en indirectas cuando se hace que el gradiente de la función objetivo se iguale a cero. Las técnicas enumerativas buscan punto a punto recorriendo todo el espacio de búsqueda. Las técnicas aleatorias también buscan punto a punto pero utilizando información adicional. Las técnicas aleatorias se subdividen en enfriamiento simulado (*simulated annealing*), en algoritmos evolutivos y en búsqueda tabú [Reeves, 1993].

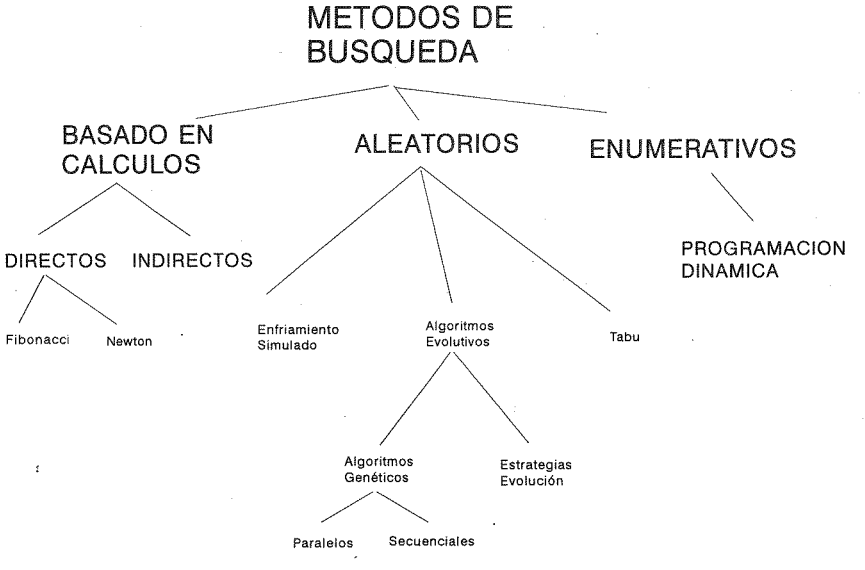

Figura 1. Taxonomía de los métodos de búsqueda.

El enfriamiento simulado es un método iterativo consistiendo en pasos de prueba y selección. Empieza con una configuración inicial del problema a optimizar, genera un nuevo estado de la configuración y lo acepta o rechaza de acuerdo a un criterio: si la función energía E (que coincide con la función objetivo que se va a minimizar) decrece, se acepta el nuevo estado, pero si aumenta, se acepta con una probabilidad $e^{-\Delta E/T}$, donde ΔE es la variación de la función energía y T es un parámetro

de control llamado temperatura. Cuanto mayor sea T, mayor será la probabilidad de aceptar el nuevo estado. Durante el proceso de optimización T se hace disminuir de valor, aceptando cada vez con menor probabilidad aquellos estados que aumenten la función energía.

La búsqueda tabú se basa en la recopilación y explotación de los principios aplicados en la resolución de problemas inteligentes. Un elemento fundamental es el uso de memoria flexible. La memoria materializa el proceso dual de crear y explotar estructuras, las cuales evolucionan condicionadas a parámetros como la frecuencia, la calidad, la influencia y la consistencia.

Los algoritmos evolutivos buscan la solución del problema guiados por los principios de selección natural. Se subdividen en estrategias de evolución y en algoritmos genéticos. La estrategia de evolución se basan en procesos de mutación y selección. Se generan mutantes de estados del espacio de configuraciones y se decide si se mantienen para la siguiente iteración (generación) si mejoran al estado anterior [Hoffmeister y Bäck, 1990]. Los algoritmos genéticos son métodos de búsqueda estocásticos inspirados en la analogía de evolución y población genética. Debido al uso de conceptos como selección, reproducción, mutación, generación, etc., se ha adoptado el nombre de algoritmo genético. El término algoritmo genético aparece acuñado por vez primera en la tesis doctoral de Bagley [1967] pero hasta que Holland [1975] publica su libro «Adaptación en sistemas naturales y artificiales», no surge un importante grupo de investigadores exclusivamente dedicado a lo que hoy día constituye el campo de desarrollo y aplicación de los algoritmos genéticos.

En general, un problema de optimización consta de:
- un espacio de búsqueda Σ con M símbolos y tamaño M^N. Así, cualquier punto de este espacio puede ser representado por un vector de N componentes de los M símbolos.
- una función objetivo o función coste F: $\Sigma \rightarrow R$ que se quiere optimizar.

Cada vector (individuo) del espacio Σ se evalúa con la función objetivo (función *fitness*), asignándosele un valor *fitness* (este proceso se llama evaluación). Los individuos con un mayor valor *fitness* tendrán mayor probabilidad de ser elegidos como «padres» (selección) en el siguiente ciclo de proceso (generación). La generación de nuevos individuos del espacio Σ se realiza mediante operadores genéticos (reproducción).

Como primera medida se genera una población inicial de individuos totalmente aleatoria. Para una mayor diversidad se puede elegir cada individuo como el mejor de n individuos aleatorios, constituyendo lo que se llama inicialización aleatoria extendida [Bramlette, 1991].

La población inicial es evaluada siguiendo un criterio que indique la calidad de cada individuo para resolver el problema. La función de evaluación es el principal enlace entre el algoritmo genético y el problema que se está resolviendo. El factor más importante en el éxito de aplicar el algoritmo genético a un problema real es la efectividad y eficiencia de la función evaluación que se tome. Debe procurarse que la función evaluación sea similar, si no igual, a la función coste que se quiera optimizar.

El algoritmo genético estándar se puede expresar en pseudocódigo como sigue:

```
Genera aleatoriamente la población inicial de individuos: P(0);
generación = 0;
Mientras (número_generaciones ≤ máximo_número_generaciones)
  Hacer {
          Evaluación;
          Selección;
          Reproducción;
          generación++;
  }
Mostrar resultados;
```

Las principales ventajas de los algoritmos genéticos son: es suficiente con un pequeño conocimiento del problema, no necesitan el control directo del entorno, es decir, no necesitan conocer datos auxiliares sobre el problema que se trata de resolver, y tampoco necesitan recorrer todo el espacio de búsqueda. La búsqueda en el espacio de solución se realiza de forma probabilística y paralela, lo que los hace más eficientes que los algoritmos enfriamiento simulado [Davis, 1987]. Aunque lo que ciertamente dota de validez a los algoritmos genéticos es su robustez, es decir, una vez que el algoritmo genético es capaz de resolver eficientemente un problema con unos parámetros determinados, aunque se modifiquen los valores de éstos parámetros, el algoritmo sigue encontrando de forma eficiente la solución del problema.

No siempre está justificado el uso de los algoritmos genéticos, sobre todo en aquellos casos en los que las técnicas existentes sean eficientes. La experimentación ha demostrado que los algoritmos genéticos deben ser utilizados para resolver aquellos problemas cuya complejidad no permita una solución directa. Problemas especialmente investigados son los problemas no resolubles polinomialmente: los problemas NP-duros [Garey y Johnson, 1979].

También se pueden utilizar algoritmos genéticos híbridos, que combinan las mejores características de los algoritmos genéticos con técnicas específicas del

problema. El algoritmo genético realiza una búsqueda gruesa y posteriormente las técnicas específicas realizan el ajuste fino de la solución.

Supóngase que se desea resolver un determinado problema: el problema del enrutamiento, el cual está siendo actualmente fuertemente investigado debido a su aplicación al guiado, centralizado o descentralizado, del tráfico en las ciudades, a redes de telecomunicación, en el contexto de redes de conmutación de paquetes, y a redes de computadores. Este problema consiste en encontrar el camino más corto entre cualquier nodo fuente, s, y cualquier nodo destino, d, de una red R. La red R={N,A} está compuesta de un conjunto finito, no vacío, de nodos N y una colección de pares ordenados de nodos diferentes, A, pertenecientes a N. Cada par de nodos (i,j) se llama arco y tiene asociado un peso $w_{i,j}$, el cual es entero, positivo e independiente de los demás nodos de la red. Además, para cualesquiera dos nodos de la red se verifica que $w_{i,j} \neq w_{j,i}$. La solución del problema del enrutamiento es una lista ordenada de nodos $<n_\pi(1) \ldots n_\pi(q)>$ que minimice la siguiente ecuación:

$$\sum_{p=1}^{q-1} W_{p,p+1} \qquad (1)$$

Para aplicar algoritmos genéticos a este problema, en primer lugar, se ha de elegir una buena codificación. La velocidad y la correcta ejecución del problema están estrechamente relacionadas con la codificación. Se ha optado por una estructura diploide (dos cadenas de caracteres) llamada individuo (figura 2). La primera cadena tiene todos los nodos de la red en cualquier orden, excepto el primer y último campo que corresponden al nodo fuente y al nodo destino respectivamente. La segunda cadena tiene el mismo número de campos que la primera y está formada por dígitos binarios 0 y 1, donde 0 indica que el nodo correspondiente de la misma posición de la primera cadena no pertenece al camino, mientras 1 significa que sí pertenece al camino [Marín y col., 1994a]. La red utilizada en los experimentos se compone de 12 nodos (figura 3), incompletamente especificada y con pesos bidireccionales. La función *fitness* se hace coincidir con la ecuación (1).

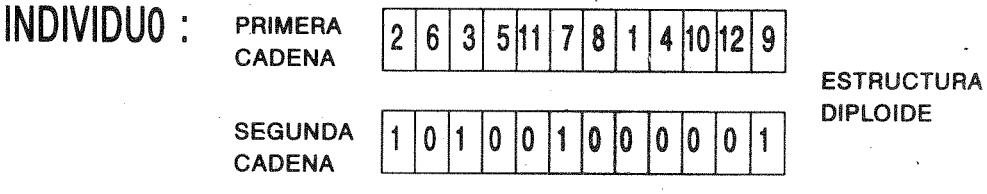

Figura 2. Codificación del problema del enrutamiento mediante individuos con estructura diploide. Camino óptimo desde el nodo 2 al nodo 9.

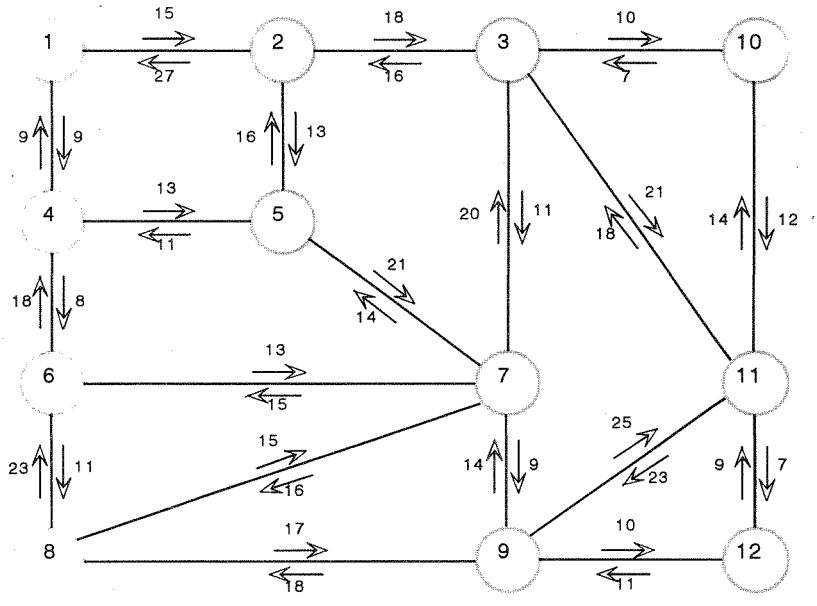

Figura 3. Red bajo estudio en el problema del enrutamiento.

En concordancia con los sistemas biológicos, cada componente del individuo representa un gen en la codificación, al valor de ese gen se le llama alelo y a su posición en la cadena, locus. Al individuo se le llama cromosoma o genotipo y a su significado en relación al problema que se trate, fenotipo.

3. OPERADORES Y SU SIGNIFICADO

Una vez generada la población inicial y asignado a cada individuo su valor *fitness*, se aplica el operador selección. En primer lugar se calcula el cociente entre el valor *fitness* de un individuo y la suma total de los valores *fitness* de todos los individuos de la población. Este resultado mide la probabilidad de selección $p_s(i)$ de cada individuo:

$$p_S(i) = \frac{f(i)}{\sum_{i=1}^{N} f(i)} \qquad (2)$$

Empezando desde la población P(t) de N individuos, se obtiene una nueva población P(t+1) aplicando N veces el operador selección. Los individuos se seleccionan de una especie de rueda de ruleta (figura 4), donde cada individuo tiene asignado un trozo en proporción a su probabilidad de selección p_s [Goldberg, 1989]. Este mecanismo de selección puede presentar problemas de convergencia prematura. La convergencia prematura está causada por la aparición de un individuo que es mucho mejor que los otros de la población aunque está lejos del óptimo; las copias de este individuo puede dominar rápidamente a la población, no pudiendo escapar de este mínimo local. Son necesarios mecanismos que aseguren mantener la diversidad en la población; la idea más generalizada es colocar censura o filtros en la selección, como por ejemplo que la inserción de un hijo en la población sólo se realiza si éste es genotípicamente diferente de todos los miembros de la población o de un número determinado de genotipos. Otra forma, bastante utilizada, de evitar la convergencia prematura es mediante el algoritmo genético «*steady-state*» propuesto por Whitley [1989] en el que no se reemplaza la población entera, sólo los individuos de menor valor *fitness* de la población son reemplazados por los nuevos hijos. También Goldberg [1987] propone la idea de valores compartidos introduciendo nichos en el espacio de búsqueda, consiguiendo un equilibrio estocástico estable.

Nº	CADENA	FITNESS	% DEL TOTAL
1	01101	169	14,4
2	11000	576	49,2
3	01000	64	5,5
4	10011	361	30,9
TOTAL		1170	100,0

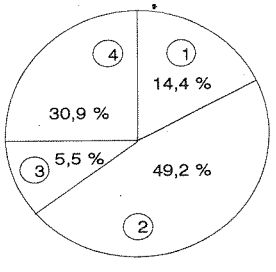

Figura 4. Operador de selección proporcional al *fitness*.

Los operadores genéticos utilizados durante la fase de reproducción son fundamentalmente el operador cruce y el operador mutación. El operador cruce se aplica en dos pasos: en el primero, los individuos se aparean (se seleccionan dos a dos) aleatoriamente con una determinada probabilidad, llamada probabilidad de cruce p_c; en el segundo paso, a cada par de individuos seleccionados anteriormente se le aplica un intercambio en su contenido desde una posición aleatoria K hasta el final, con $K \varepsilon [1, m-1]$, siendo m la longitud del individuo. K se llama punto de cruce

y determina la subdivisión de cada padre en dos partes que se intercambian para formar dos nuevos hijos según se puede ver en la figura 5. Esto se conoce como cruce ordinario o cruce de un punto.

Existen muchas variantes del operador cruce diseñadas para problemas específicos. Se puede destacar el operador cruce uniforme en el que cada alelo se selecciona aleatoriamente de uno u otro padre con probabilidad uniforme obteniéndose mayor diversidad en la población [Eshelman y Schaffer, 1991]. El objetivo del operador cruce es recombinar subcadenas de forma eficiente, esta gestión recibe el nombre de construcción de bloques.

El operador mutación consiste en la alteración aleatoria de cada uno de los genes del individuo con una probabilidad de mutación p_m, como se puede ver en la figura 6. El objetivo de la mutación es producir diversidad en la población. Supongamos que al generar aleatoriamente la población inicial o después de varias generaciones, en la misma posición de todos los cromosomas sólo aparece un único elemento del alfabeto utilizado, esto supondrá que con los operadores reproducción y cruce nunca cambiarán dicho elemento, por lo que puede ocurrir que jamás se alcance la solución más óptima a nuestro problema. Es necesario un operador que pueda cambiar aleatoriamente cualquier elemento de cualquier individuo. Por otro lado, la probabilidad de aparición del operador de mutación no debe ser grande para no perjudicar la correcta construcción de bloques.

El operador mutación origina variaciones elementales en la población y garantiza que cualquier punto del espacio de búsqueda pueda ser alcanzado. Es importante decir que la inexperiencia hace que se intente contrarrestar la convergencia prematura aumentando la probabilidad de mutación, pero esto presenta efectos secundarios, ya que una alta probabilidad de mutación devalúa el papel del cruce en la construcción de bloques e impulsa al algoritmo genético a realizar una búsqueda exhaustiva en el espacio de soluciones. Es necesario balancear la explotación de buenas soluciones con la exploración en regiones desconocidas hasta ese momento.

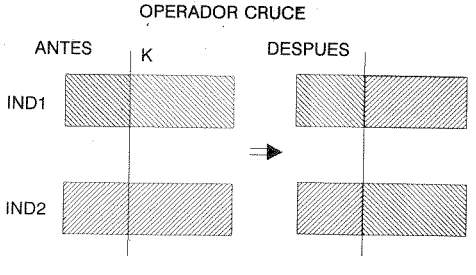

Figura 5. Operador cruce ordinario o de un sólo punto.

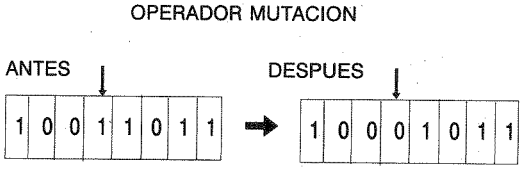

Figura 6. Operador mutación.

Existen muchas variantes del operador mutación, entre las que podemos destacar la mutación exponencial cuando decrece la diversidad de la población [Forgaty, 1989] o la mutación adaptativa que es eliminada cuando afecta negativamente al rendimiento del individuo. Yao [1993] introduce el concepto de entropía para medir la diversidad de la población.

El rendimiento de los algoritmos genéticos depende de los parámetros: tamaño de la población, frecuencia de aplicación de los operadores cruce y selección, además de los mecanismos utilizados para la selección y el cruce. En la figura 7 se muestran los resultados obtenidos para la optimización de una función polinómica [Marín y col., 1992]. Se utiliza una codificación binaria, los valores *fitness* están normalizados, la población es de 100 individuos, la longitud del individuo es 30 (intervalo de estudio $[0, 2^{30}]$) y las probabilidades de cruce y mutación son 0.5 y 0.03, respectivamente. Se ha representado en abscisas el número de generaciones y en ordenadas el promedio de los valores *fitness*. A partir de la séptima generación se ha conseguido una estabilización en el promedio *fitness*, lo que asegura que los parámetros iniciales son los adecuados para un algoritmo genético estándar, esto es, un número de individuos grande, probabilidad de cruce sobre 0.5 y una probabilidad de mutación muy baja, aproximadamente como la inversa del tamaño de los cromosomas. Estos valores fueron propuestos por De Jong [1975] como los más adecuados en problemas de optimización.

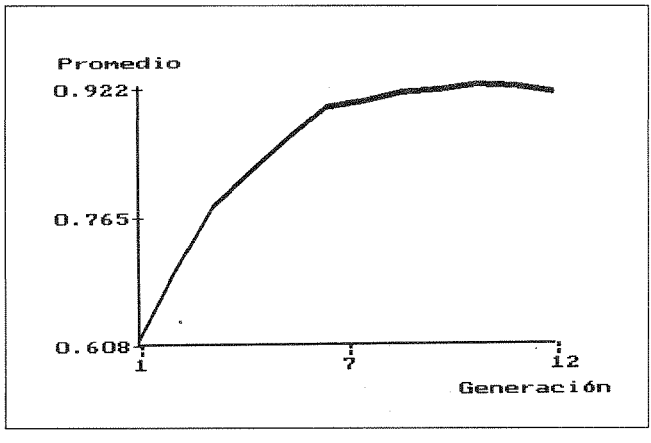

Figura 7. Promedio de los valores *fitness* normalizados frente al número de generaciones. Comportamiento correcto.

A continuación, se va a ver como afecta la modificación de los parámetros iniciales anteriores en el promedio de los valores *fitness*. En la figura 8 se mantienen los valores óptimos en las probabilidades de cruce y mutación, pero el número de individuos es bajo, 10, y el tamaño de los cromosomas es pequeño, también 10 (intervalo de estudio $[0, 2^{10}]$). El comportamiento del algoritmo no es el adecuado

puesto que al ser el número de individuos y su tamaño muy pequeño hace que el efecto de la mutación sea muy apreciable. Cuando la mutación afecte a un bit de los más significativos de un individuo, hace que el promedio *fitness* disminuya drásticamente, obteniéndose bastantes oscilaciones.

En la figura 9 se muestra otro mal comportamiento del algoritmo genético debido a que se ha mantenido el número de individuos, 10, y el tamaño de los cromosomas, 10, y, además, la probabilidad de cruce es pequeña (0.3). Se observa cómo la baja probabilidad de cruce hace que aunque empiece evolucionando correctamente, llegando en las generaciones intermedias a una buena aproximación de la solución, en las siguientes generaciones el promedio *fitness* no se estabiliza.

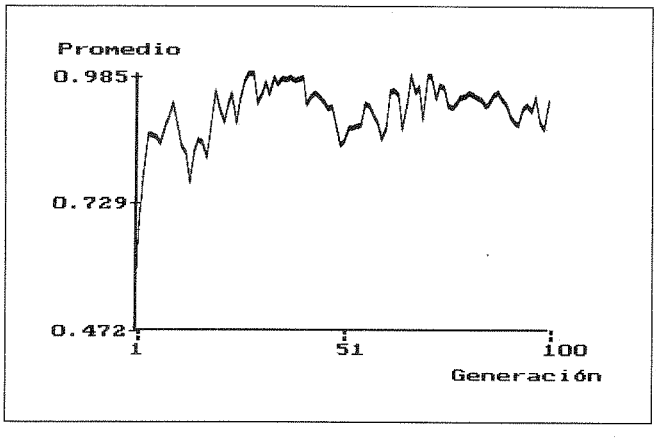

Figura 8. Promedio de los valores *fitness* normalizados frente al número de generaciones. Comportamiento incorrecto debido a altas tasas de mutación.

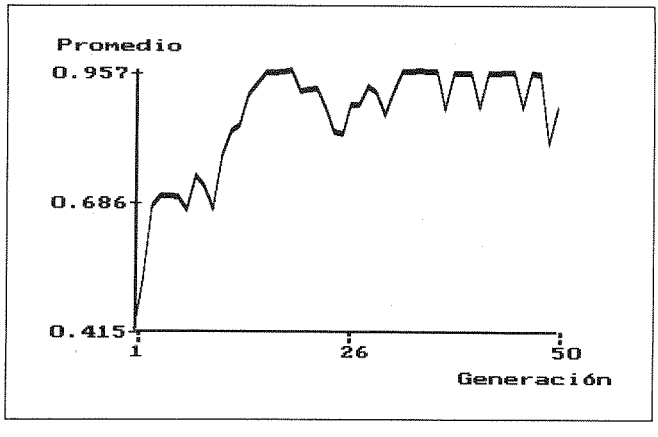

Figura 9. Promedio de los valores *fitness* normalizados frente al número de generaciones. Comportamiento incorrecto debido a la baja probabilidad de cruce.

Para la resolución del problema del enrutamiento se utiliza el operador de selección muestreo estocástico universal [Baker, 1987], el operador de cruce parcialmente mapeado (PMX) [Goldberg, 1987], un operador de permutación que intercambia aleatoriamente las posiciones de dos nodos de la primera cadena y un nuevo operador, llamado desbloqueo, que cada cierto número de generaciones regenera aquellos individuos de la población que causan que el algoritmo genético no funcione apropiadamente [Marín y col., 1994a]. La figura 10 muestra la frecuencia con la que el algoritmo genético encuentra caminos óptimos, válidos y no válidos para nodos seleccionados aleatoriamente. Hay 5 vehículos en ruta, por lo que los pesos de la red son modificados a través del tiempo, con lo que el algoritmo genético debe trabajar de forma dinámica para conseguir un control adaptativo.

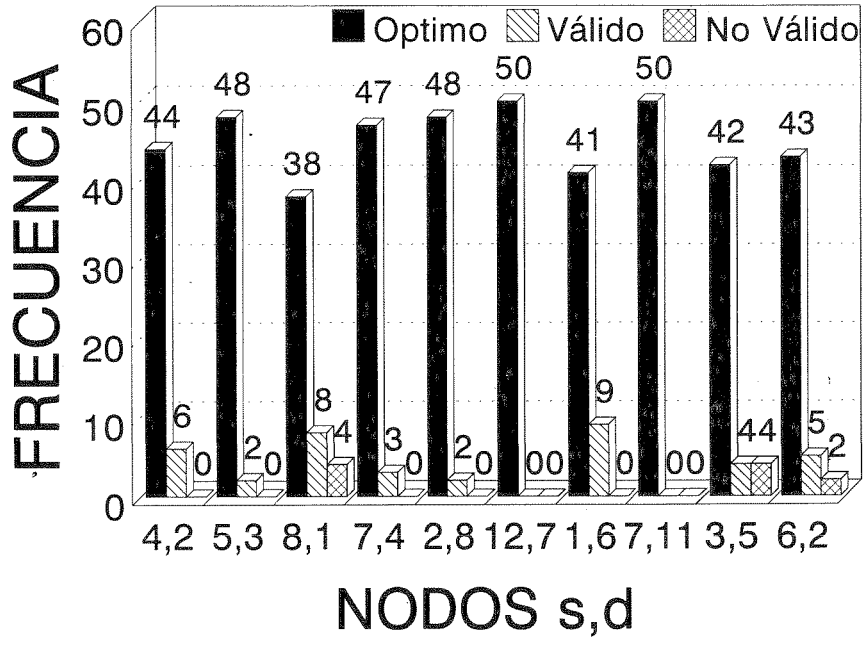

Figura 10. Caminos óptimos, válidos y no válidos desde el nodo s al nodo d con cinco vehículos en ruta.

4. FUNDAMENTOS TEÓRICOS DE LOS ALGORITMOS GENÉTICOS

Una vez vistos los operadores básicos y su implementación, cabría plantearse el interrogante de si los nuevos hijos realizan una búsqueda más óptima que los padres, es decir, si el A.G. realiza una adecuada construcción de bloques. Esta sección trata de dotar al algoritmo genético de rigurosidad matemática. La herramienta de trabajo utilizada es el esquema.

Dado el alfabeto inicial Ω, se trabaja con un alfabeto ampliado Ω' formado por el anterior más un nuevo elemento: el metasímbolo '*', que indica que en las posiciones que ocupe puede haber cualquier elemento del alfabeto inicial Ω. Por ejemplo, si el alfabeto es binario $\Omega = \{0,1\}$ se forma un nuevo alfabeto $\Omega' = \{0,1,*\}$. A todas las cadenas formadas por elementos de Ω' se les llama **esquemas**.

El número de posibles esquemas viene dado por:

$$(\text{base} + \text{metasimbolo})^{\text{longitud individuo}} \tag{3}$$

Dado un individuo, si se reemplazan r genes por el metasímbolo * (con $0 \le r \le m$), siendo m la longitud del individuo, se crea un esquema para el que este individuo es un representante. El número total de esquemas que tiene como representante a un determinado individuo, aplicando el teorema binomial, es:

$$\sum_{r=0}^{m} \binom{m}{r} = \binom{m}{0} + \binom{m}{1} + \ldots + \binom{m}{m} = 2^m \tag{4}$$

Cualquier individuo es un representante de 2^m esquemas, lo que implica que el número de esquemas muestreados en una población de N individuos no debe ser mayor de $N.2^m$. La evaluación de un individuo da información sobre un gran número de esquemas. Holand [1975] le dio a este resultado el nombre de paralelismo implícito y Goldberg [1987] estimó que el número de esquemas procesados útilmente es del orden de N^3.

Para fijar conceptos, se considera un cromosoma binario con tan sólo tres posiciones. Así, el espacio de búsqueda es tridimensional, se puede representar por un cubo (figura 11) y existen $(2+1)^3=27$ esquemas distintos. Los puntos 000, 001, 010, 011 son los vértices del plano F del cubo. Estos puntos pueden ser caracterizados como todos los puntos cuya primera posición sea un 0 y las demás inespecificadas ($F \rightarrow 0^{**}$, en la figura 11 con trazo más grueso).

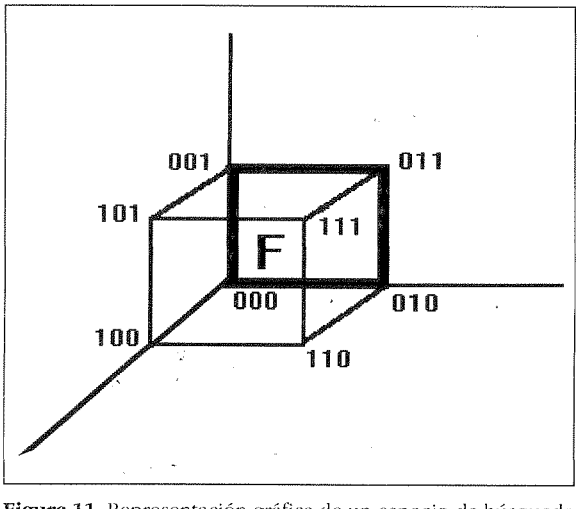

Figura 11. Representación gráfica de un espacio de búsqueda tridimensional.

En general, cada esquema corresponde a un hiperplano en el espacio de búsqueda. El esquema *** corresponde a todo el cubo y representa el espacio de búsqueda completo.

Se define el **orden** de un esquema E, y se representa O(E), como el número de posiciones fijas que hay en él. Así, O(10**11) = 4 y O(1*****) = 1. Se define la **longitud** de un esquema E, y se representa por L(E), como la distancia entre la primera y la última posición definida del esquema. Así, L(10**11) = 6 - 1 = 5 y L(1*****) = 1 - 1 = 0. También se define el *fitness* de un esquema como el *fitness* medio del conjunto de individuos que estén representados en dicho esquema. Una buena construcción de bloques corresponderá a los esquemas con longitud y orden bajo y con alto valor *fitness*.

En lo que sigue, se quiere encontrar el número de representantes de un esquema E que sobreviven en la siguiente generación. Se supone que en la generación actual, t, hay S representantes de un esquema E, esto es, S(E,t), y se quiere calcular el número de representantes de dicho esquema en la siguiente generación (t+1), esto es, S(E,t+1).

4.1. Efecto de la selección en un esquema E

En la primera fase del algoritmo genético los cromosomas que tienen mayor probabilidad de ser seleccionados para la reproducción son aquellos que tienen un valor *fitness* más alto. Por tanto, el número de representantes de un esquema que sobreviven al operador selección es directamente proporcional al valor *fitness* de cada uno de ellos; esto se cuantifica como que la probabilidad de seleccionar un cromosoma perteneciente al esquema E viene dada por $\bar{f}(E)/\bar{f}$, siendo $\bar{f}(E)$ el valor *fitness* del esquema E y $\bar{f}$ el valor medio de los valores *fitness* de todos los individuos de la población, luego:

$$S(E,t+1) = S(E,t).\frac{\bar{f}(E)}{\bar{f}} \qquad (5)$$

4.2. Efecto del cruce en un esquema E

En la siguiente fase del algoritmo genético se aplica el operador cruce a los individuos seleccionados. Sean estos $A=a_1a_2a_3a_4a_5$ y $B=b_1b_2b_3b_4b_5$. Suponiendo que A es un representante del esquema E caben las siguientes posibilidades:

a) Si A = B es obvio que después del cruce los nuevos individuos A' y B' serán representantes de los mismos esquemas. No se destruye ningún esquema.

b) Si el esquema E está formado únicamente por metasímbolos, no será destruido nunca.

c) Si el esquema E sólo tiene una posición definida es equivalente a decir que el orden del esquema es O(E) = 0, y nunca será destruido puesto que sea cual sea la posición k del cruce, al menos uno de los nuevos individuos A' o B' e incluso ambos siguen siendo representantes del esquema E.

d) Si la longitud del esquema es 1, esto es, el esquema sólo tiene dos posiciones especificadas y además son contiguas, la única posibilidad de que se destruya ese esquema será cuando la posición k del cruce caiga entre ambas posiciones especificadas. La probabilidad de destrucción del esquema E será el cociente entre los casos favorables, L(E) = 1, y los casos posibles, m-1, ya que k ε [1, m-1].

e) Si la longitud del esquema es L(E) = 2, es decir, cuando entre la primera y última posición especificada hay un metasímbolo, en analogía con el anterior apartado, el esquema E será destruido cuando la posición de cruce K caiga entre las posiciones especificadas. Por tanto, la probabilidad de destrucción viene dada por el cociente entre la longitud del esquema L(E) y las posibles posiciones del cruce m-1.

En general, la probabilidad de que un esquema sea destruido en la fase de cruce es L(E)/(m-1). Como el operador cruce no actúa siempre, sino que lo hace con una cierta probabilidad que se ha llamado probabilidad de cruce p_c, se puede ampliar la expresión anterior a p_c.L(E)/(m-1).

Según lo anterior, la probabilidad de supervivencia p_s es igual a:

$$p_S = 1 - p_d = 1 - \frac{p_C . L(E)}{m-1} \qquad (6)$$

Hay que aclarar que esta expresión se ha obtenido en el caso más desfavorable, es decir, cuando B no comparte ninguna de las posiciones especificadas del esquema, por lo que la probabilidad de supervivencia del esquema se debe expresar como:

$$p_S >= 1 - \frac{p_C . L(E)}{m-1} \qquad (7)$$

4.3. Efecto de la mutación en un esquema E

En la tercera y última fase del algoritmo genético se aplica el operador mutación. Como hemos visto, la mutación consiste en la alteración aleatoria de cada uno de los genes del individuo con probabilidad p_m. Para que un esquema sobreviva es necesario que todas sus posiciones especificadas sobrevivan. La probabilidad de que un gen sobreviva es $1-p_m$. Puesto que cada mutación es estadísticamente independiente, un esquema sobrevive cuando cada una de sus posiciones fijas O(E) sobrevive. Multiplicando O(E) veces la probabilidad de supervivencia de un gen, $1-P_m$, se tiene la probabilidad de supervivencia del esquema en la mutación $(1-p_m)^{O(E)}$. Como la probabilidad de mutación es muy baja, $p_m << 1$, la expresión anterior se puede aproximar por $1 - O(E).p_m$

Combinando los efectos de los tres operadores considerados independientes entre ellos y después de despreciar pequeños términos producto, el número de representantes de un esquema que sobreviven a la siguiente generación viene dado por la expresión:

$$S(E,t+1) \geq S(E,t).\frac{\bar{f}(E)}{\bar{f}}\left(1-\frac{p_c.L(E)}{m-1}-O(E).p_m\right) \qquad (8)$$

Esta ecuación se conoce como **teorema fundamental de los algoritmos genéticos o teorema de los esquemas**.

Es obligado notar que se producen errores de muestreo, es decir evaluaciones incorrectas del *fitness* del esquema, haciéndose cada vez mayores cuanto más converja la población hacia subespacios del espacio de búsqueda.

5. ALGORITMOS GENÉTICOS PARALELOS

La aplicación de los algoritmos genéticos a problemas con un gran espacio de búsqueda, con costosas funciones de evaluación y utilizando grandes tamaños de población, hace necesaria la realización de implementaciones lo más rápidas posibles. El camino natural es la paralelización del algoritmo genético [East y Macfarlane, 1993].

El algoritmo genético no es fuertemente paralelizable debido a la necesidad de control global tanto en el proceso de selección como en la aplicación del operador cruce. Para solventar este problema son necesarias iteraciones no-locales entre los procesadores, lo que conlleva altos costes de comunicación. Se han propuesto varias aproximaciones para distribuir la carga computacional, las cuales pueden ser agrupadas en tres clases generales: migración, difusión y *farming*.

5.1. Migración

En este modelo, la población total se divide en subpoblaciones de igual tamaño, asignando un procesador para cada subpoblación. Periódicamente cada procesador selecciona sus mejores individuos y los envía a sus procesadores más cercanos, recibiendo copias de los mejores individuos de sus procesadores vecinos que reemplazarán a los peores individuos de su propia subpoblación (esto se conoce como migración de individuos).

Según la forma de intercomunicación entre las subpoblaciones, se tienen distintos tipos de algoritmos genéticos distribuidos. Entre estos, se pueden distinguir:

a) Algoritmos genéticos en estrella: La subpoblación con mayor promedio *fitness* es seleccionada como maestra y las demás como esclavas. Todas las subpoblaciones esclavas migran sus mejores individuos a la maestra, y, a su vez, ésta migra sus mejores individuos a cada una de las esclavas (figura 12.a).

b) Algoritmos genéticos en red: En este modelo todas las subpoblaciones migran sus mejores individuos a todas las demás (figura 12.b).

c) Algoritmos genéticos en anillo: Cada subpoblación envía sus mejores individuos a la subpoblación vecina más próxima en un único sentido de flujo (figura 12.c).

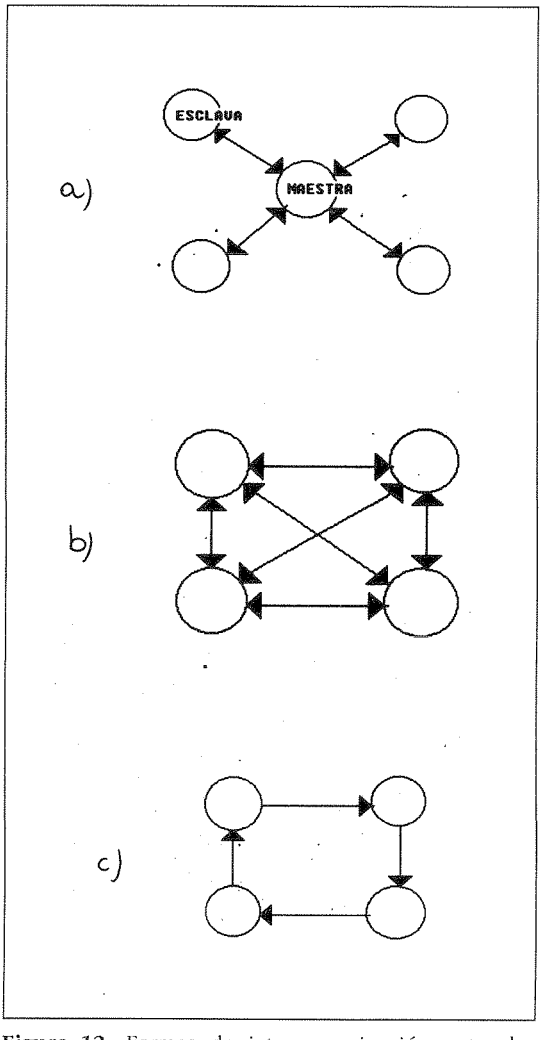

Figura 12. Formas de intercomunicación entre las subpoblaciones. a) en estrella, b) en red y c) en anillo.

Dependiendo del modo de intercambio, se habla de modelo «*island*» si los individuos migran a subpoblaciones arbitrarias, normalmente elegidas al azar, y modelo «*stepping-stone*» si la migración se limita sólo a los vecinos más próximos.

Para la ejecución del modelo migración se deben definir cuáles son los procesadores vecinos, la frecuencia de intercambio (tasa de migración), el número de individuos intercambiados y la clase de búsqueda que va a realizar cada procesador. El paralelismo inherente no está completamente explotado por no conocerse a priori los valores más adecuados de estos parámetros.

Las primeras implementaciones fueron realizadas por Tanese [1987] y por Pettey y Leuze [1987] corriendo en máquinas MIMD, típicamente *transputers*.

5.2. Difusión

En este modelo, sólo evoluciona una población, se asigna cada individuo de la población en una celda de una rejilla plana. La selección y el cruce se realizan sólo entre individuos más próximos en la rejilla de acuerdo a una estructura predefinida. Este modelo es una aproximación de grano fino (*fine-grained*) y hace una mayor exploración del espacio de búsqueda debido a su mecanismo de selección local, evitando una sobre-selección. Esta mayor exploración va acompañada de una carga computacional extra, por lo que sólo es recomendable para aquellos problemas que necesiten una fuerte exploración del espacio de búsqueda. Se puede correr de forma síncrona [Collins y Jefferson, 1991] o de forma asíncrona en máquinas paralelas de memoria compartida [Maruyama y col., 1992].

Se ha medido el tiempo de complejidad del modelo difusión y del algoritmo genético estándar [Spiessens y Manderick, 1991]. En la tabla 1 se reproducen los resultados obtenidos, siendo n el tamaño de la población, s el tamaño de los vecinos más próximos y l la longitud del genotipo. Presenta dos tiempos de complejidad porque utilizó varios métodos de selección (selección proporcional, *ranking*, t*ournament local*, ...). Se obtiene que el A.G. estándar necesita un tiempo de ejecución polinomial frente al tiempo de ejecución lineal del modelo difusión.

5.3. *Farming*

Se selecciona a un procesador como maestro que gestiona a la población total selecciona a los mejores individuos, que envía a los demás procesadores (procesadores esclavos). Los procesadores esclavos recombinan estos individuos para

crear hijos y los evalúan antes de enviarlos de nuevo al procesador maestro [Davis, 1991].

Se ha realizado una estrategia basada en una mezcla de migración y *farming* [Marín y col., 1994b] usando una agrupación de estaciones de trabajo corriendo el software de dominio público PVM 3.1 (*Parallel Virtual Machine*) [Sunderam, 1990]. La idea básica de la aproximación es que una estación de trabajo maestro distribuye la población entre las estaciones de trabajo esclavas, las cuales periódicamente envían sus mejores individuos al maestro para que los clasifique y redistribuya a los esclavos. Esta estrategia se ha aplicado al problema del enrutamiento; en la figura 13 se muestra el rendimiento frente al número de generaciones para un único servidor (secuencial), y para 2 y 3 servidores, obteniendo que después de dos intervalos de comunicaciones (número de generaciones en las que cada servidor envía sus mejores individuos al maestro) el rendimiento mejora con el número de servidores. En la figura 14 se representa el *speed-up* (cociente entre el tiempo de ejecución secuencial y el paralelo) frente al número de máquinas esclavas tomando como parámetro el intervalo de comunicaciones. Se han representado los peores casos para intervalos de comunicación de 1, 3 y 5 generaciones. Se aprecia la eficiencia de la aproximación utilizada.

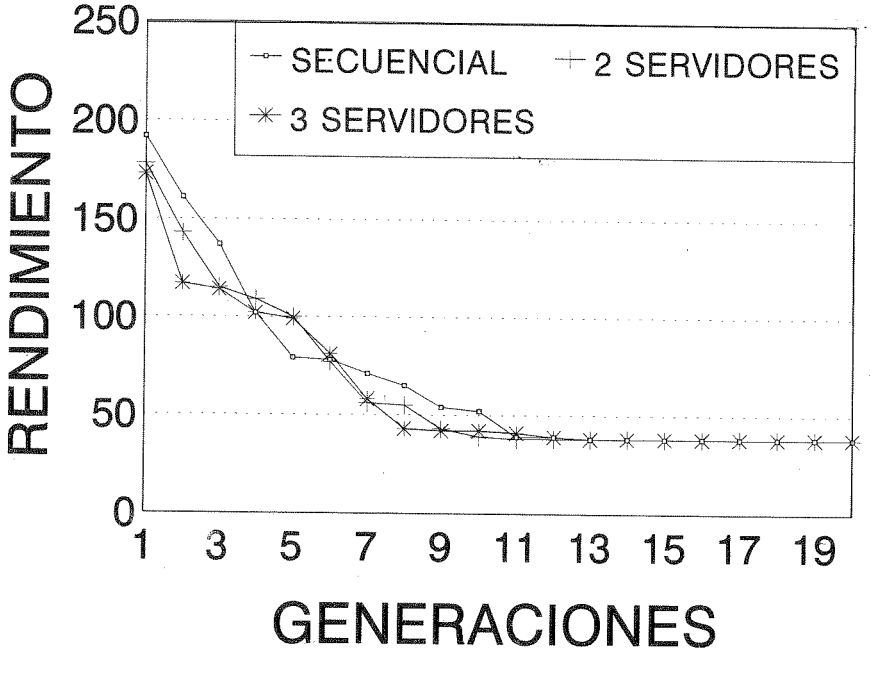

Figura 13. Rendimiento frente al número de generaciones para distintos servidores.

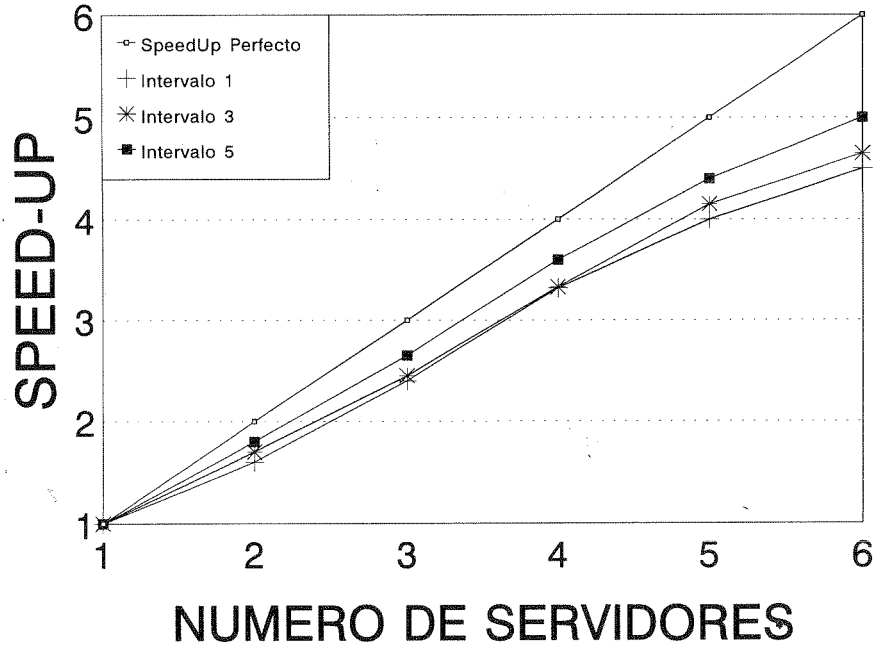

Figura 14. *Speed-Up* frente al número de servidores para distintos intervalos de comunicaciones.

6. DISEÑO DE REDES NEURONALES

La aplicación del algoritmo genético al diseño de redes neuronales para resolver una determinada tarea está siendo dirigido desde tres campos de aplicación [Yao, 1992]:

a) En la búsqueda del conjunto óptimo de pesos de conexión para una red en la que se ha fijado el número de neuronas y la conectividad entre ellas. El algoritmo genético hace el papel de regla de aprendizaje.

b) En la búsqueda de la óptima topología (número de neuronas y conectividad) una vez fijada la regla de aprendizaje.

c) En la búsqueda de la óptima regla de aprendizaje, ajustando los parámetros de aprendizaje y de la función de evaluación. Previamente se ha seleccionado el número de neuronas y la conectividad entre ellas.

6.1. Síntesis de los pesos de conexión

El diseñador ha fijado la topología de la red neuronal (número de neuronas de entrada y salida, número de capas ocultas, número de neuronas en cada capa oculta y la conectividad entre todas las neuronas) y el algoritmo genético debe encontrar el conjunto óptimo de pesos, esto es, realiza el aprendizaje de la red neuronal. Se aplica a redes neuronales *feed-forward*, cambiando el clásico algoritmo de aprendizaje *back-propagation* [Rumelhart y col., 1986] por el algoritmo genético.

Inicialmente se generan aleatoriamente todos los individuos de la población, donde cada uno de ellos consiste en la concatenación de los pesos de la red, los cuales codifican los pesos de conexión (genotipo). El genotipo se decodifica asociándole una red neuronal (fenotipo) con pesos fijos (en este caso, el fenotipo es igual al genotipo). Se presentan los pares de entrenamiento entrada-salida calculando el error cuadrático medio entre la salida ideal y la obtenida al evaluar la red (este valor es el valor *fitness* del individuo). Se aplican los operadores genéticos selección, cruce y mutación para obtener una nueva generación. Estos pasos se repiten hasta que se minimicen los errores (figura 15).

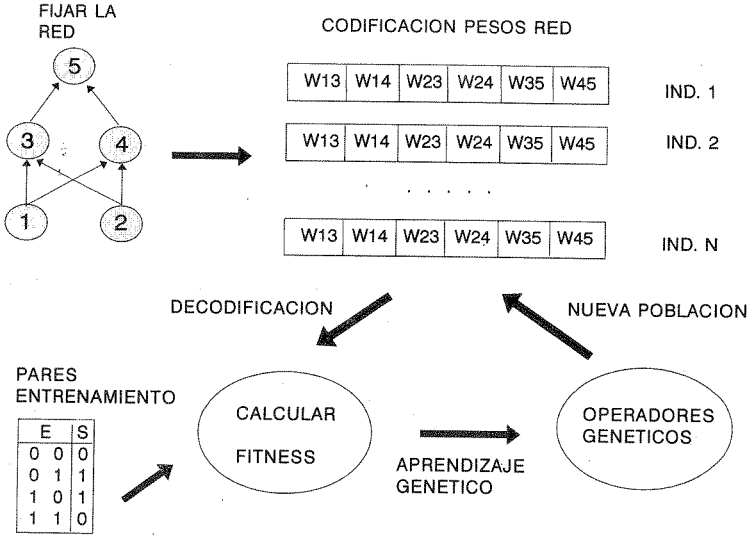

Figura 15. Diagrama de flujo en la síntesis de los pesos de conexión.

La codificación de los pesos de la red puede ser mediante una cadena binaria de cierta longitud. Por ejemplo, Whitley, Starkweather y Bogart [Whitley y col., 1993] en su algoritmo llamado GENITOR utilizan 8 bits para representar cada peso, siendo una magnitud con signo comprendida entre +127 y -127 con el cero apareciendo dos veces. Previamente se ha fijado el intervalo sobre el que se mueven los pesos. Lo aplicaron a X-OR de 2 bits, a un codificador 4-2-4 y a sumadores de dos bits. El principal inconveniente de la codificación binaria es la limitación de la precisión por la discretización de los pesos. Si se codifica con muchos bits para aumentar la precisión, los individuos se hacen muy grandes aumentando considerablemente el tiempo de entrenamiento de la red. La alternativa es hacer un buen balance entre el número de bits de cada peso de conexión y el intervalo de codificación utilizando técnicas de codificación dinámica [Schraudolph y Belew, 1992]. Típicamente, se utiliza codificación Gray para asegurar que cambios pequeños en los bits corresponda a cambios pequeños en los valores de los parámetros.

También se ha propuesto una codificación con números reales pero los operadores genéticos clásicos no pueden ser aplicados directamente. Montana y Davis [1989] diseñan operadores genéticos específicos que incorporan heurística utilizada en el entrenamiento de redes neuronales, obteniendo mayor rapidez en el entrenamiento que con el *back-propagation*. Lo aplica a problemas de reconocimiento de señales acústicas en entornos ruidosos (bajo agua).

6.2. Síntesis de la topología

La decisión de elegir la topología de la red se basa en previas experiencias del diseñador o en procesos de prueba y error. Los algoritmos constructivos/destructivos dependen fuertemente de la topología inicial y pueden caer en mínimos locales, no garantizando una topología óptima o presentando convergencias demasiado rápidas o demasiado lentas.

El espacio de búsqueda es muy grande, y el número de posibles nodos y conexiones es a priori ilimitado. Topologías bastante diferentes pueden tener capacidades similares (espacio de búsqueda multimodal) y topologías muy parecidas pueden tener diferentes rendimientos (espacio de búsqueda de apariencia engañosa). Estas características hacen factible la aplicación del algoritmo genético a la síntesis de redes neuronales.

Inicialmente se generan aleatoriamente todos los individuos de la población, correspondiendo cada uno de ellos a una determinada topología de la red (fenotipo). El genotipo se decodifica asociándole una red neuronal (fenotipo). Se presentan los pares de patrones de entrenamiento entrada-salida calculando el error cuadrático

medio entre la salida ideal y la obtenida al evaluar la red. El entrenamiento se realiza con algoritmos clásicos, normalmente el *back-propagation*. El valor *fitness* del individuo se toma como la ponderación del error cuadrático medio más otros términos que penalicen la complejidad de la topología de la red, el tiempo de entrenamiento, etc. Se aplican los operadores genéticos selección, cruce y mutación para obtener una nueva generación. Estos pasos se repiten hasta que se minimicen los errores (figura 16).

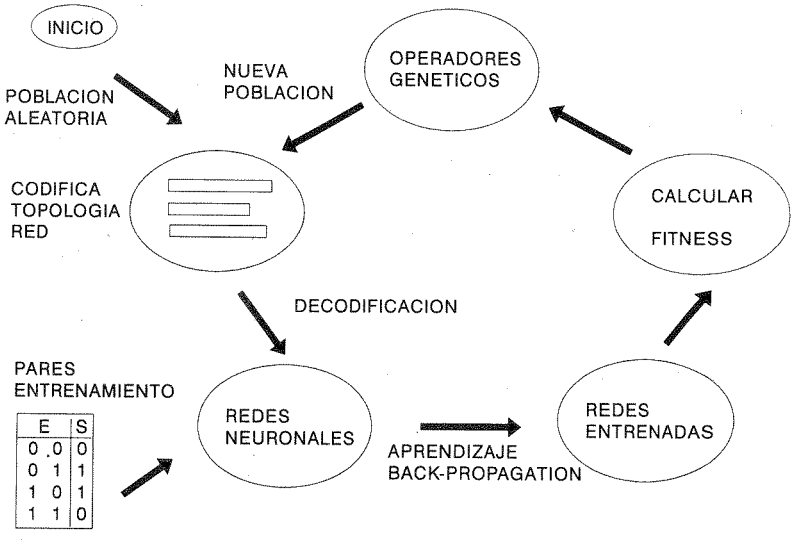

Figura 16. Diagrama de flujo en la síntesis de la topología.

La codificación directa representa explícitamente cada conexión de la red mediante dígitos binarios. En la figura 17 se muestra una hipotética codificación para una red que resuelve el problema de la X-OR de dos bits [Miller y col., 1989]. En general, para una red con N neuronas se forma una matriz de conexión **W** de orden NxN donde sus elementos $w_{i,j}$ indican conexión desde la neurona i a la neurona j. Los elementos $w_{i,j} \in [0, 1]$, representando el 1 que existe dicha conexión y el 0 que no existe. Si la matriz de conexiones es triangular inferior, la red sólo tiene conexiones hacia adelante (*feed-forward*) y si es triangular superior con elementos distintos de cero en la diagonal principal, representa que es *feed-back* con auto-realimentación (tipo red recurrente). La ventaja de esta codificación es la fácil aplicación de los operadores genéticos para la reducción de conexiones (podaje de la red), pero en casos de grandes redes son necesarias enormes matrices de conexión, con lo cual el tiempo de proceso se hace casi prohibitivo.

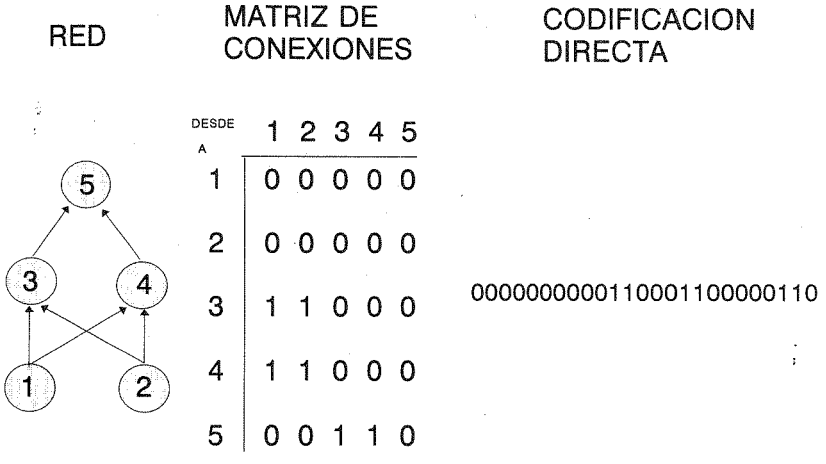

Figura 17. Codificación directa de una red XOR de dos bits.

Para reducir el tamaño del individuo (genotipo), se realiza una codificación indirecta en la que sólo aparecen las características más importantes de la conectividad. Los detalles de la matriz de conexión se solucionan con reglas de desarrollo [Kitano, 1990], proyecciones [Harp y col., 1989] o grado de densidad [Dodd y col., 1991] durante la decodificación del individuo (fenotipo). Aquí, el fenotipo es distinto al genotipo. La gran ventaja de la codificación indirecta es la compactación.

En la codificación indirecta, los pesos de la red se codifican en una cadena de dígitos binarios [Harp y col., 1990], los cuales son de longitud variable y están constituidos por una concatenación de áreas. Cada área se subdivide en áreas de especificación de los parámetros y área de especificación de las proyecciones o conectividad. Esta codificación requiere utilizar operadores genéticos específicos, ya que individuos funcionalmente correctos pueden producir hijos no funcionales (por ejemplo, hijos sin conexiones hacia la capa de salida de la red, etc.). Su sistema llamado Neurogenesys tiene un filtro para desechar individuos con anormalidades o purificar individuos que presenten pequeñas anormalidades. Todo el estudio se aplica para el diseño de redes *feed-forward*, en aproximación de la función seno, el problema de la X-OR de dos bits y en reconocimiento de dígitos de tamaño 4x8. Sus resultados fueron sorprendentes: el reconocimiento de dígitos eliminó todas las neuronas ocultas, es un problema linealmente separable, y en la X-OR, permitiendo conexiones directas desde la capa de entrada a la capa de salida, eliminó una neurona.

Otro método de codificación indirecta utiliza descripciones recursivas de redes, con producción gramatical para generar matrices de conexión [Kitano, 1990]. Este sistema de producción de reglas funciona correctamente si el número de neuronas ocultas se incrementa mientras que las soluciones de codificación explícitas del algoritmo genético se degradan. Esta representación es compacta, eliminando la redundancia permutacional. Otro método de codificación indirecta [Gruau, 1992] empieza con una neurona, hace crecer a la red y la controla mediante un programa estructurado en árbol. Puede expresar redes recursivas añadiendo bifurcación condicional y permite colecciones de programas etiquetados que pueden ser invocados como una operación. Presenta las importantes propiedades de compactación y escalabilidad. También Mjolsness, Sharp y Alpert [Mjolsness y col., 1989] utilizan una codificación compacta donde la matriz de conexiones se construye a partir de un patrón inicial por medio de recursividad de operadores duplicación, introduciendo en su sistema la propiedad de escalabilidad.

Redes equivalentes funcionalmente que tienen diferente topología, realizan la misma tarea al mismo nivel de rendimiento, produciéndose redundancia permutacional [Radcliffe, 1993]. Esto implica un enorme aumento en el espacio de búsqueda. Una red *feed-forward* con una única capa oculta de h unidades, tiene una redundancia potencial de h!. El número exacto parece depender del esquema de codificación utilizado.

6.3. Síntesis de la regla de aprendizaje

La estructura de la red está fuertemente ligada a la existencia de un más o menos potente algoritmo de aprendizaje, no se puede garantizar que el proceso de aprendizaje de la red sea el más idóneo para un determinado algoritmo de aprendizaje y, por otra parte, si el proceso de aprendizaje no da buenos resultados, no es posible saber si ha fallado la red o el algoritmo de aprendizaje. La relación entre la topología de la red y el proceso de aprendizaje es generalmente desconocida.

Inicialmente se generan aleatoriamente todos los individuos de la población, cada uno de ellos codifica la regla de aprendizaje (fenotipo). La regla de aprendizaje es la misma para todas las conexiones de la red. El genotipo se decodifica en una regla de aprendizaje (fenotipo). Se presentan los pares de patrones de entrenamiento entrada-salida calculando el error cuadrático medio entre la salida ideal y la obtenida al evaluar la red. El *fitness* se toma como la ponderación del error cuadrático medio más otros términos que penalicen la complejidad de la topología de la red, el tiempo de entrenamiento, etc. Se aplican los operadores genéticos selección, cruce y mutación para obtener una nueva generación. Estos pasos se repiten hasta que se minimicen los errores (figura 18).

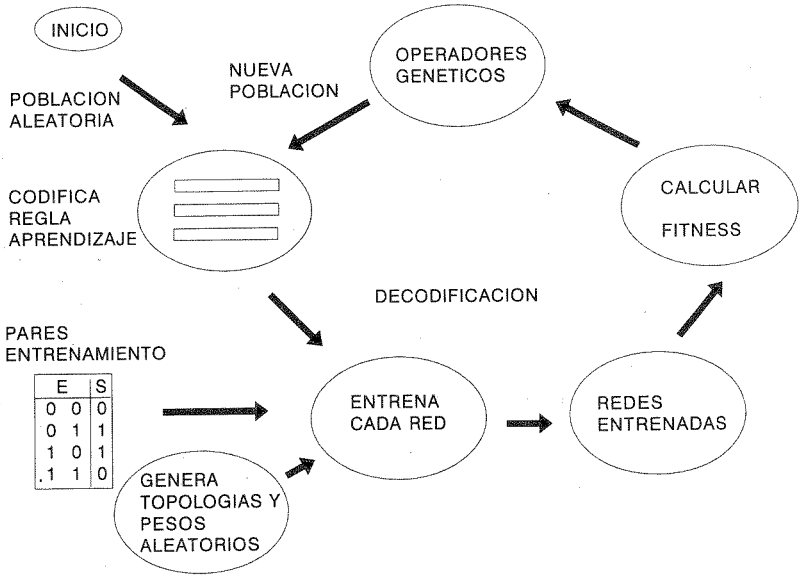

Figura 18. Diagrama de flujo en la síntesis de la regla de aprendizaje.

Una regla de aprendizaje se puede describir como una función lineal de n variables y sus productos:

$$\Delta w(t) = \sum_{k=1}^{n} \sum_{i_1,i_2,\ldots,i_k=1}^{n} \left[\theta_{i_1,i_2,\ldots,i_k} \prod_{j=1}^{k} x_{i_j}(t-1) \right] \qquad (9)$$

donde t es el tiempo, Δw es la variación del peso, x_k (k=1,...,n) son las variables y θs son coeficientes reales. La optimización de la regla de aprendizaje es obtener los coeficientes θs.

Chalmers [1990] y posteriormente Fontanari y Meir [1991] definen la forma de la regla de aprendizaje como una función lineal de variables locales (sólo necesitan información de esa conexión pero no de las demás conexiones) y términos producto. Cada individuo es una cadena binaria que codifica exponencialmente 10 coeficientes más un parámetro de escala. Después de 1000 generaciones, se llega a la conocida regla de aprendizaje delta [Widrow y Hoff, 1960] y algunas de sus variantes [Chalmers, 1990].

6.4. Síntesis de la topología y los pesos de conexión

No existe mucha investigación en el diseño de redes neuronales donde varios o todos los campos anteriores hayan sido combinados. La mayoría de ellas se centran en un único campo y con redes *feed-forward*, utilizando el algoritmo de aprendizaje *back-propagation*. Esas redes son apropiadas para problemas combinatoriales (las salidas son función de las entradas pero no dependen de los estados anteriores de las neuronas), pero no pueden resolver problemas secuenciales (las salidas son también función de los estados anteriores). Se pretende que el algoritmo genético interaccione en los campos de investigación anteriores, esto es, debe encontrar la topología idónea de la red con los valores de los pesos de conexión y utilizando también al algoritmo genético como regla de aprendizaje [Marín y Sandoval, 1993].

6.4.1. Esquema de representación de la red

El esquema debe ser capaz de englobar a redes *feed-forward* para tratar problemas combinatoriales (la matriz de pesos **W** es triangular inferior con ceros en la diagonal principal) y a una red *feed-back* para tratar problemas secuenciales (**W** puede tener elementos distintos de cero en cualquier fila o columna). Así, el esquema está completamente abierto a cualquier tipo de topología. Se asemeja a redes recurrentes [Almeida, 1987, 1988; Pineda, 1987, 1988] de tiempo discreto [Williams y Zipser, 1989]. La elección del paradigma que se debe utilizar para resolver un determinado problema es completamente transparente para el diseñador. Su única tarea es adaptar la función *fitness* para premiar o castigar la clase de conectividad seleccionada [Marín y Sandoval, 1993].

Cada individuo se representa por una cadena de caracteres binarios con dos áreas independientes:

1) El área de definición de la conectividad (ACD), que incluye los siguientes campos (figura 19):

   - Neuronas de entrada (I).
   - Neuronas de salida (O).
   - Número de capas ocultas (L).
   - Neuronas ocultas en cada capa(H_l).
   - Tipo de conectividad (μ)
   - Número de bits de codificación de los pesos (W).

2) El área del espacio de los pesos (AWS).

Los campos I, O, H_l y W están codificados con 3 bits pudiendo tomar como máximo el valor 8 (2^3), el campo L está codificado con 2 bits y el campo μ con un

único bit, representando μ=0 conectividad *feed-forward* y μ=1 total conectividad (figura 20). Dependiendo de los valores del área ACD se reserva dinámicamente memoria para el área AWS.

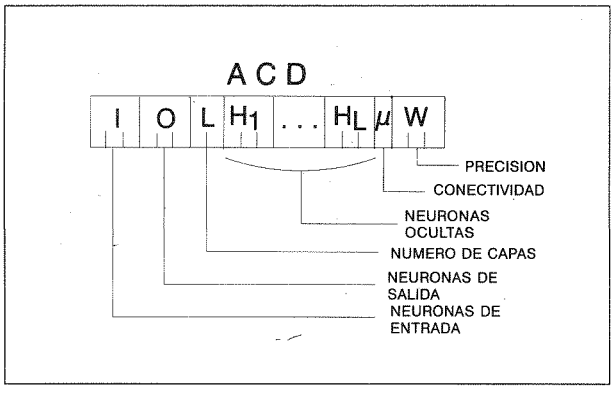

Figura 19. Esquema de representación de una red neuronal. Área de definición de la conectividad.

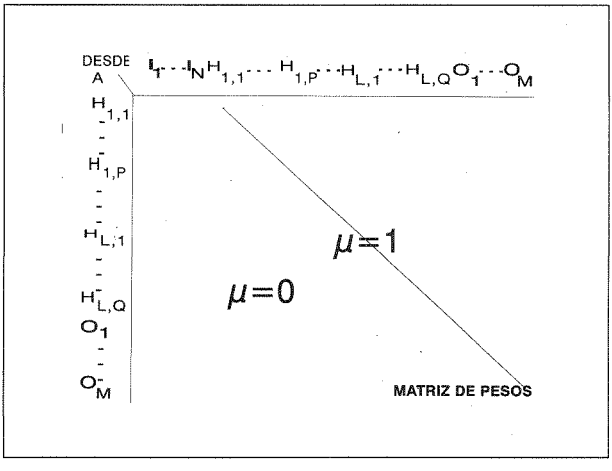

Figura 20. Matriz de pesos de una red neuronal. Tipo de conectividad.

Antes de la codificación de la red, los campos de las neuronas de entrada y salida de la red se definen de acuerdo con el problema que se vaya a resolver. El A.G. empieza generando una población aleatoria de individuos que codifican a una red específica. Esto es, cada individuo de la población representa una posible arquitectura donde los valores de los pesos son obtenidos aleatoriamente. La figura 21 muestra la codificación de una hipotética red que tiene 5 neuronas de entrada, 2 neuronas de salida, 2 capas ocultas con 4 neuronas en la primera capa y 3 en la segunda capa,

con total conectividad *feed-forward* y nula conectividad *feed-back* con los pesos codificados con 5 bits. Obsérvese la reserva dinámica de pesos. Cuando se da conectividad *feed-back*, el sistema considera que no se puede hablar de capas ocultas y las considera como neuronas en una única capa.

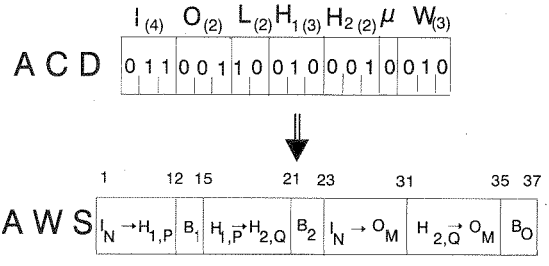

Figura 21. Ejemplo de codificación.

Los individuos (genotipos) son evaluados de acuerdo a una medida cuantitativa de su validez como solución al problema tratado. Esta evaluación se lleva a cabo mediante la función *fitness*, la cual consta de tres términos: un primer término que penaliza el error entre la solución deseada y la solución obtenida en cada momento, $\theta(E)=\alpha.E$, y es específico para cada aplicación; otro término que castiga un gran número de neuronas ocultas, $\zeta(H) = \beta.\Sigma H_i$ y un último término que penaliza los valores altos de los pesos (incluyendo el *bias* si el ε_B se ha tomado como cero) $\eta(W) = \delta.(\Sigma \text{ abs}(\textit{pesos}) + \Sigma \text{ abs}(\textit{bias}))$:

$$\textbf{\textit{FITNESS}} = \theta(E) + \zeta(H) + \eta(W) \qquad (10)$$

Las constantes de proporcionalidad α, β y δ se eligen de acuerdo con el grado de penalización que se desee para cada término. Los individuos con el *fitness* más bajo son aquellos que tienen una mayor probabilidad de ser seleccionados para la reproducción. Los operadores cruce y mutación son los encargados de guiar la búsqueda de la solución más óptima de la red, eliminando o colocando conexiones y/o neuronas. El operador cruce trabaja independientemente para cada área del esquema de representación.

Se utiliza una codificación binaria empaquetada del genotipo, el mecanismo de selección es el muestreo estocástico universal de Baker [Baker, 1987] y el operador cruce es el de dos puntos, proporcionando una buena efectividad en cuanto a diversidad y máxima robustez. Se ha optado por mantener al mejor individuo de la población para la siguiente generación (elitismo).

El algoritmo genético realiza de forma simultánea el aprendizaje de la red, obteniendo el conjunto óptimo de pesos de conexión y la optimización de la red reduciendo el número de neuronas ocultas y el número de conexiones entre neuronas. La figura 22 muestra el diagrama de flujo del sistema. Se han llevado a cabo

diversos experimentos tanto de problemas combinatoriales, con conexiones *feed-forward*, como problemas secuenciales, con conexiones *feed-back*. Dos de ellos se muestran en esta sección: el problema combinatorial de la paridad de tres bits y el problema secuencial de la detección de una determinada secuencia, en concreto de la secuencia 110.

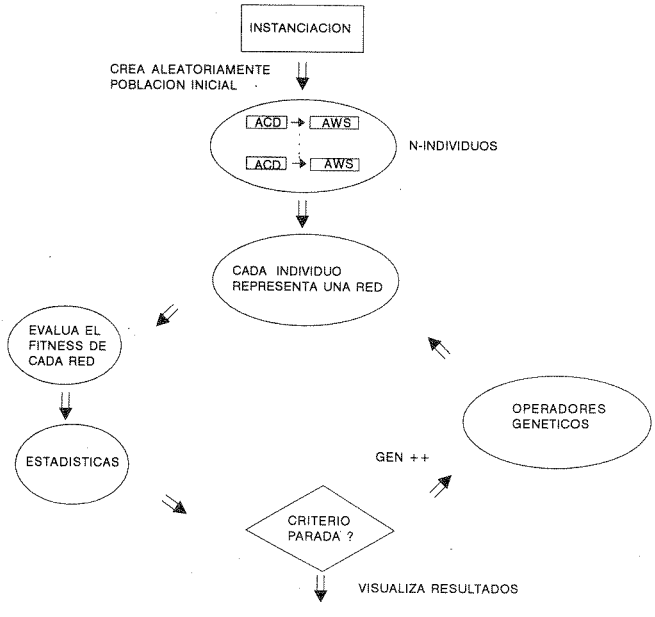

Figura 22. Diagrama de flujo de la síntesis de los pesos y la topología.

El problema combinatorial de la paridad de N-bits es fácilmente resuelto mediante una red multicapa *feed-forward* utilizando un algoritmo de aprendizaje tipo *back-propagation*. Ha sido ampliamente estudiado el mínimo número de capas ocultas, y dentro de éstas el mínimo número de neuronas de cada capa, que son necesarias para resolver la paridad de N-bits; eligiendo una apropiada función de transferencia que contemple ciertas restricciones conocidas a priori se puede reducir el número de neuronas necesarias en la capa oculta y acelerar el aprendizaje con respecto a aquellas que utilicen una función de transferencia tradicional, tal como la función sigmoide: el problema de la paridad de N-bits no puede ser resuelto usando dos neuronas ocultas tipo sigmoide cuando $N > 2$, mientras que tomando una apropiada función de transferencia puede ser resuelto con sólo dos neuronas ocultas [Stork y Allen, 1992].

Se quiere que el algoritmo genético haga evolucionar a la red en un tiempo sustancialmente corto hasta ser capaz de resolver el problema de la paridad de 3-bits con una única neurona oculta sigmoidal siempre y cuando se permitan conexiones desde la capa de entrada a la capa de salida. Se permite total conectividad *feed-forward* pero la conectividad *feed-back* es completamente eliminada (μ=0); como función de activación de todas las neuronas se utiliza la función sigmoide y la función *fitness* encargada de guiar al algoritmo genético viene dada por la ecuación (10).

El área AWS está codificado como un vector de 14 dígitos (número de pesos incluyendo *bias*) de tipo doble de siete bits cada uno, tomando valores en el intervalo [-1, 1) (2^7 posibles valores). El término $\theta(E)$ de la función *fitness* es en este caso el error cuadrático total entre la salida obtenida, $y_j(K)$, y la salida deseada, $d_j(K)$, de la K-ésima neurona de salida multiplicada por una constante de proporcionalidad α: $\theta(E)= \alpha.\Sigma_K\Sigma_j(Y_j(K)-d_j(K))^2$. Las constantes de proporcionalidad α, β y δ se han seleccionado de tal forma que las soluciones de la función completen una línea recta. Así, se deben penalizar los valores altos de los pesos y los *bias* (la conexión entre neuronas se elimina cuando su peso correspondiente es cero), y, como se mueven dentro del intervalo [-1, 1), como máximo será $\eta(W+bias)=14$, siendo δ=1 (notar que con esto se realiza el podaje de la red neuronal); también se debe penalizar y en mayor grado el número de neuronas de la capa oculta, que en nuestro ejemplo es como máximo dos, por lo que el valor de la constante de proporcionalidad β se toma como 15, haciendo $\zeta(H)=30$ como máximo; pero lo que realmente nos interesa es que la red opere sin fallos (la salida obtenida debe coincidir con la salida deseada) por lo que la constante de proporcionalidad α ha de ser mayor que $\beta+\delta$, en nuestro caso se ha tomado como 45.

En la figura 23 se muestra la topología de la red incluyendo los valores de los pesos una vez que la red ha aprendido. Se observa cómo se han eliminado todas las conexiones a y desde una de las neuronas de la capa oculta y los dos *bias* de la capa oculta.

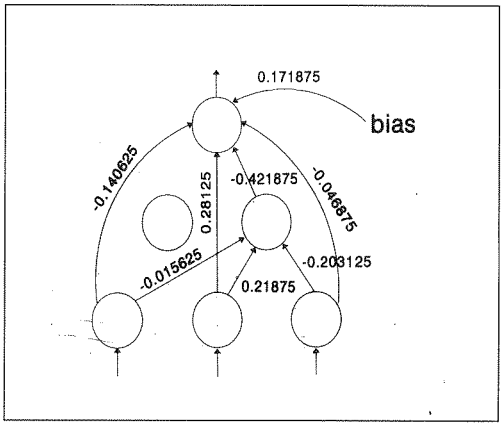

Figura 23. Red que resuelve el problema de la paridad de tres bits.

Los experimentos llevados a cabo han puesto de manifiesto que el parámetro pendiente p de la función sigmoide juega un papel fundamental en la consecución de una correcta solución. La figura 24 muestra la dependencia logarítmica de la

generación frente al parámetro p con los valores de los pesos en el intervalo [-1, 1). En línea continua se puede ver el tiempo (medido en número de generaciones) necesario para que la red alcance una solución (fase de aprendizaje) y en línea discontinua el tiempo necesario para que la red, una vez aprendida, elimine una neurona oculta (fase de optimización). Los distintos valores de p modifican la rapidez con que el A.G. es capaz de lograr el aprendizaje y la optimización. Al aumentar el valor de p la red aprende más rápidamente pero, sin embargo, la optimización es más lenta, mientras que cuando p disminuye el aprendizaje es más lento y la optimización más rápida. Para valores mayores que un cierto umbral (mayores de 0.01) el algoritmo genético no encuentra una solución al problema. Todos los experimentos se han ejecutado diez veces, seleccionando el promedio de los mejores individuos de la población.

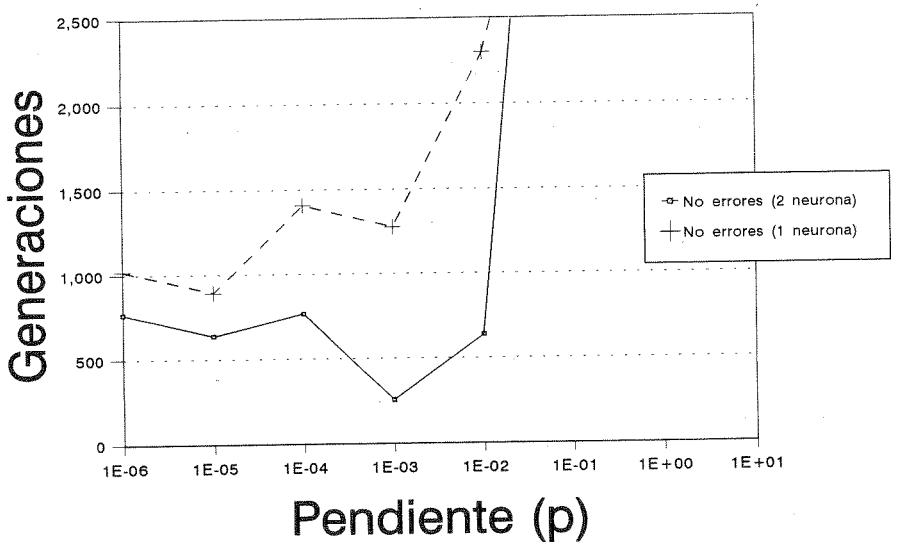

Figura 24. Evolución del número de generaciones con la pendiente de la función sigmoide. La línea continua indica el aprendizaje y la línea discontinua la optimización de la red.

La detección de secuencias es un típico problema secuencial en el que a una entrada van llegando en serie bits aleatorios y la red debe ser capaz de detectar (poniendo su salida a 1) un patrón específico o secuencia; en caso contrario, la salida permanecerá a cero. Se permite solapamiento. Se quiere detectar la secuencia 110 como se muestra en el siguiente ejemplo.

Entrada 10110111010111011010 . . .
Salida 00001000100000100100 0 . . .

La topología inicial de la red neuronal se muestra en la figura 25.a. Se trata de una red neuronal recurrente en tiempo discreto completamente conectada pudiendo tener como máximo cuatro neuronas (a priori no se sabe el número de neuronas necesarios, tomando cuatro por similitud con el número de biestables necesarios para resolver el problema), donde todas ellas son neuronas de entrada y sólo una de ellas es, además, de salida.

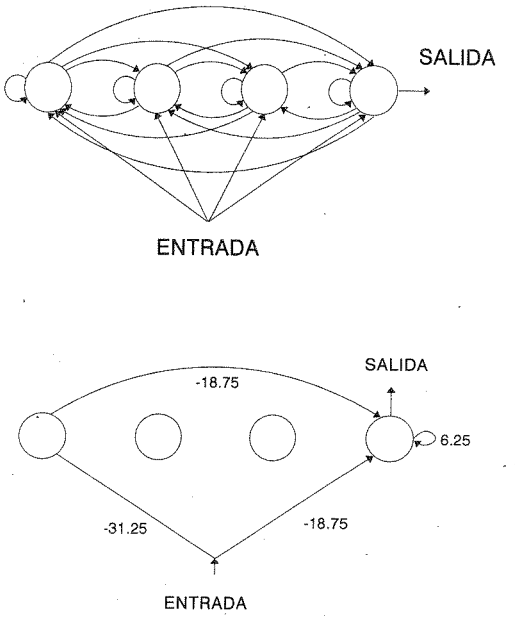

Figura 25. Detector de la secuencia 110. a) red inicial y b) red final optimizada. Se han eliminado 2 neuronas y 16 conexiones.

El área AWS está codificado como un vector de 20 dígitos de tipo doble de cinco bits cada uno, tomando valores en el intervalo [-100, 100) permitiendo total conectividad tanto *feed-forward* como *feed-back*. Cada evaluación consiste en comparar los resultados de la red con los pesos a evaluar para una entrada de 50 bits aleatorios con los resultados teóricos que debería producir, contabilizando el número de errores producidos multiplicado por una constante α, correspondiendo esto con el valor de $\theta(E)$. Al igual que en el caso anterior, las constantes α, β y δ se han tomado como 10^4, 10^3 y 1, respectivamente, para que se penalice en mayor grado el número de errores producidos, después el número de neuronas y por último el número de conexiones entre las neuronas. La función $\zeta(H)$ de la función *fitness* penaliza en este caso el número de conexiones en vez del número de neuronas.

La figura 25.b muestra la topología resultante, incluyendo los valores de los pesos, después de ejecutar el A.G.. Se han eliminado dos neuronas y 16 conexiones, resultando una red con sólo dos neuronas y 4 conexiones.

La figura 26 muestra el rendimiento (valor de la función *fitness*) frente al número de evaluaciones, representando la línea continua al mejor individuo de la población y la línea discontinua la media de la población sobre 10 ejecuciones. Se observa, que en un tiempo muy corto (sobre 5000 evaluaciones) se consigue que la red elimine

los errores para posteriormente optimizar el número de conexiones (elimina una neurona sobre la evaluación 85000 y dos neuronas sobre la evaluación 235000). Se han realizado experimentos con los valores de los pesos en distintos intervalos, obteniéndose los mejores resultados en el mayor intervalo considerado [-100, 100).

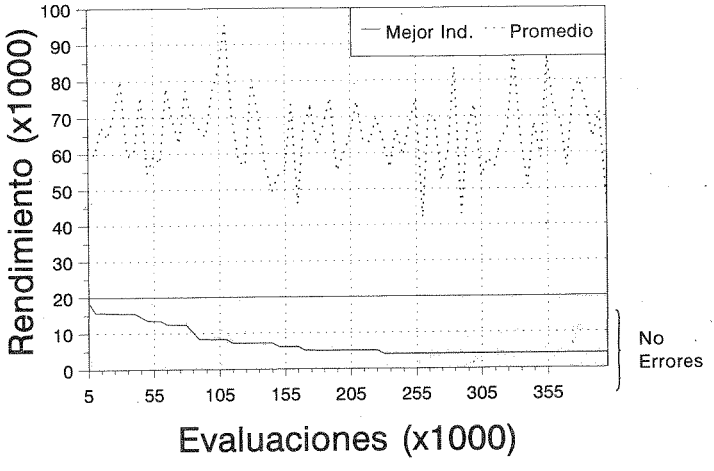

Figura 26. Rendimiento frente al número de evaluaciones. La línea continua representa al mejor individuo y la línea discontinua al promedio de la población.

7. CONCLUSIONES

Una área de investigación, totalmente abierta, es la utilización de algoritmos genéticos, o cualquier otra técnica evolutiva, en el proceso de automatización del diseño de redes neuronales artificiales. Se puede aplicar el algoritmo genético para todo el diseño de una red neuronal, desde la especificación de los valores de los pesos de conexión, hasta encontrar la óptima topología de la red, e incluso, en la búsqueda de la óptima regla de aprendizaje. Otra ventaja es que el diseñador no se debe preocupar del tipo de conexionado ni del paradigma que tiene que utilizar.

Se demuestra una buena eficiencia cuando se aplica a la resolución de pequeños problemas. Para problemas de mayor complejidad, el tamaño de los individuos crece enormemente, haciéndose prohibitivo en tiempo computacional el esquema de codificación directa. Es necesaria una mayor investigación en métodos de codificación más compactos y escalables, además de dotar al algoritmo genético de mayor velocidad de convergencia haciéndolo correr en máquinas paralelas.

Se apunta, cada vez con mayor frecuencia, a la utilización del algoritmo genético para realizar la búsqueda global de la óptima solución a un determinado problema, esto es, localizar regiones óptimas, y después aplicar métodos específicos para ese problema que realicen una búsqueda local, seguramente, con mayor precisión y rapidez.

AGRADECIMIENTOS

Este trabajo fue parcialmente financiado por la Comisión Interministerial de Ciencia y Tecnología (CICYT), proyecto No. TIC91-0965.

REFERENCIAS

Almeida, L., «A learning rule for asynchronous perceptrons with feedback in a combinatorial environment», *IEEE First International Conference on Neural Networks*, Vol. 2, (1987), 609-618.

Almeida, L., «Backpropagation in Perceptrons with Feedback», en Eckmiller, R. y Malsburg, C.v.d. (Eds.), *Neural Computers*, Springer-Verlag, 1989, 199-208.

Bagley, J.D., «The behavior of adaptive systems which employ genetic and correlation algorithms», (Doctoral dissertation, University of Michigan), *Dissertation Abstracts International*, 28 (12), 5106B, 1967.

Baker, J.E., «Reducing bias and inefficiency in the selection algorithm», en Grefenstette, J.J. (Ed.), *Proceedings of the Second International Conference on Genetic Algorithms*, 1987, 14-21.

Bramlette, M.F., «Initialization, mutation and selection methods in genetic algorithms for function optimization», en Belew, R.K. y Booker, L.B. (Eds.) *Proceedings of the Fourth International Joint Conf. On Genetic Algorithms*, Morgan Kaufmann Publishers, 1991, 100-107.

Chalmers, D.J., «The evolution of learning: an experiment in genetic connectionism», en Touretzky, D.S., Elman, J.L. y Hinton, G.E. (Eds.), *Proceedings of the 1990 Connectionist Models Summer School,* Morgan Kaufmann, 1990, 81-90.

Collins, R.J. y Jefferson, D.R., «Selection in massively parallel genetic algorithms», en Belew, R.K. y Booker, L.B. (Eds.), *Proceedings of the Fourth International Joint Conf. On Genetic Algorithms*, Morgan Kaufmann Publishers, 1991, 249-256.

Davis, L.D. (Ed.), *Genetic Algorithms and Simulated Annealing,* Los Altos, Morgan Kaufmann, 1987.

Davis, L.D., *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, 1991.

De Jong, K.A., «An analysis of the behavior of a class of genetic adaptive systems». (Doctoral dissertation, University of Michigan). *Dissertation Abstracts International,* 36 (10), 5140B, 1975.

Dodd, N., Macfarlane, D. y Marland, C., «Optimisation of artificial neural networks structure using genetic techniques implemented on multiple transputers», *Proceedings of Transputing '91*, (1991).

East, I.R. y Macfarlane, D., «Implementation in Occam of Parallel Genetic Algorithms on Transputer Networks», en Stender, J. (Ed.), *Parallel Genetic Algorithms*, IOS Press, 1993, 43-63.

Eshelman, L.J. y Schaffer, J.D., «Preventing premature convergence in genetic algorithms by preventing incest» en Belew, R.K. y Booker, L.B. (Eds.), *Proceedings of the Fourth International Joint Conf. On Genetic Algorithms*, Morgan Kaufmann Publishers, 1991, 115-122.

Fahlman S.E. y Lebiere, C., «The cascade-correlation learning architecture», en Touretzky, D.S. (Ed.), *Advances in Neural Information Processing Systems 2*, Morgan Kauffmann, 1990, 524-532.

Fontanari, J. F. y Meir, R., «Evolving a learning algorithm for the binary perceptron», *Networks*, Vol. 2, (1991), 353-359.

Forgaty, T.C., «Varying the probability of mutation in the genetic algorithms», en Grefenstette, J.J. (Ed.), *Proceedings of the Second International Conference on Genetic Algorithms*, 1987, 104-109.

Frean, M., «The upstart algorithm: a method for constructing and training feed-forward neural networks», *Neural Computation*, Vol. 2, (1990), 198-209.

Garey, M.R., y Johnson, D.S., *Computers and Intractability. A Guide to the Theory of NP-Completeness*, Freeman and Company, New York, 1979.

Goldberg, D.E. y Richardson, J., «Genetic algorithms with sharing for multimodal function optimization», en Grefenstette, J.J. (Ed.), *Proceedings of the Second International Conference on Genetic Algorithms*, 1987, 41-49.

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

Gruau, F.C., «Cellular encoding of genetic neural networks», *Technical Report*, LIP-IMAG Ecole Normale Superieure de Lyon, 46 Allée d'Italie 69007 Lyon, France, (1992).

Harp S.A., Samad, T. y Guha, A., «Towards the genetic synthesis of neural networks», en Schaffer, J.D. (Ed.), *Third International Conference on Genetic Algorithms*, Morgan Kauffmann, 1989, 360-369.

Hirose, Y., Yamashita, K. y Hijiya, S., «Back-propagation algorithm which varies the number of hidden units», *Neural Networks*, Vol. 4, (1991), 61-66.

Hoffmeister, F. y Bäck, T., «Genetic algorithms and evolution strategies: similarities and differences», *Lectures Notes in Computer Science*, N. 496, Springer-Verlag, (1990), 455-469.

Holland, J.H., *Adaptation in natural and artificial systems*, Ann Arbor: The University of Michigan Press, 1975.

Joya, G., Frias, J.J., Marín, M.M. y Sandoval, F., «New Learning Strategies from the Microscopy Level of an Artificial Neural Networks», *Electronics Letters*, Vol. 29, N. 20, (1993), 1775-1777.

Karnin, E.D., «A simple procedure for pruning back-propagation trained neural networks», *IEEE Transactions on Neural Networks*, Vol 1, N. 2, (1990), 239-242.

Kitano, H., «Designing neural networks using genetic algorithms with graph generation system», *Complex Systems,* Vol. 4, (1990), 461-476.

Le Cun, Y., Denker, J.S. y Solla, S.A., «Optimal brain damage», en Touretzky, D.S. (Ed.), *Advances in Neural Information Processing Systems, 2,* Morgan Kaufmann, 1990, 598-605.

Maricic, B. y Nikolov, Z., «GENNET- System for computer aided neural network design using genetic algorithms», *IEEE Proceedings International Joint Conference on Neural Network*, Vol. 1, (1990), 102-105.

Marín, F.J., García, F. y Sandoval, F., «Algoritmos genéticos: Una estrategia para la búsqueda y la optimización», *Informática y Automática,* Vol. 25, N. 3 y 4, (1992), 5-15.

Marín, F.J. y Sandoval, F., «Genetic synthesis of discrete-time recurrent neural network», en Mira, J., Cabestany, J. y Prieto, A. (Eds.), *New Trends in Neural Computation,* Springer-Verlag, 1993, 179-184.

Marín, F.J., González, F.J. y Sandoval, F., «The Routing Problem in Traffic Control using Genetic Algorithms», *2nd IFAC Symposium on Intelligent Components and Instruments for Control Applications,* (1994a).

Marín, F.J., Trelles-Salazar, O. y Sandoval, F., «Genetic Algorithms on LAN-message passing architecture using PVM: Applications to the routing problem», enviada a *International Conference On Evolutionary Computation (PPSNIII),* (1994b).

Marín, F.J., González, F.J. y Sandoval, F., «Design of optimal artificial neural networks topologies with genetic algorithms», enviada a *IEEE Transaction on Neural Networks,* (1994c).

Maruyama, T., Konagaya, A. y Konishi, K., «An Asynchronous Fine-Grained Parallel Genetic Algorithm», en Männer, R. y Manderick, B. (Eds.), *Parallel Problem Solving from Nature 2,* Elsevier Science Publishers B.V., 1992, 563-572.

McClelland, J.L., Rumelhart, D.E., y PDP Research Group, *Parallel Distributed Processing*, MIT Press, 1986.

Miller, G.F., Todd, P.M. y Hegde, S.U., « Designing Neural Networks using Genetic Algorithms», en Schaffer, J.D. (Ed.), *Third International Conference on Genetic Algorithms*, Morgan Kauffmann, 1989, 379-384.

Minsky, M. y Papert, S., *Perceptrons: An introduction to computational geometry*, Cambridge, MA, MIT Press, 1988.

Mjolsness, E., Sharp, D.H. y Alpert, B.K., «Scaling, machine learning, and genetic neural nets «, *Advances in Applied Mathematics,* Vol. 10, (1989), 137-163.

Montana, D.J. y Davis, L., «Training feedforward neural networks using genetic algorithms», *Proceedings of Eleventh International Joint Conference on Artificial Intelligent,* (1989), 762-767.

Pettey, C.B., Leuze, M.R. y Grefenstette, J.J., «A Parallel Genetic Algorithm», en Grefenstette, J.J. (Ed.), *Proceedings of the Second International Conference on Genetic Algorithms*, Lawrence Erlbaum Associates, 1987, 155-162.

Pineda, F.J., «Generalization of Back-propagation to Recurrent Neural Networks», *Physical Review Letters*, Vol. 59, (1987), 2229-2232.

Pineda, F.J., «Dynamics and Architecture for Neural Computation», *Journal of Complexity*, Vol. 4, (1988), 216-245.

Radcliffe, N.J., «Genetic set recombination and its application to neural network topology optimisation» *Neural Computing and applications,* Vol. 1, N.1, (1993), 67-90.

Robbins, G.E., Hughes, J.C., Plumbley, M.D., Fallside, F. y Prager, R., «Generation and adaptation of neural networks by evolutionary techniquees (GANNET), *Neural Computing and applications,* Vol. 1, N. 1, (1993), 22-30.

Reeves, C.R., *Modern Heuristic Techniques for Combinatorial Problems,* Blackwell Scientific Publications, 1993.

Rumelhart, D.E., Hinton, G.E. y Williams, R.J., «Learning internal representations by error propagation», *Parallel Distributed Proccessing*, Vol. 1, (1986), 310-362.

Schaffer, J.D., *Proceedings of the Third International Conf. On Genetic Algorithms,* Morgan Kaufmann Publishers, 1989.

Schraudolph, N.N., *Computer Science & Engeniering Department,* University of California, San Diego, La Jolla, CA 92093-0114.

Schraudolph, N.N. y Belew, R.K. «Dynamic parameter encoding for genetic algorithms», *Machine Learning,* Vol. 9, N. 1, (1992), 9-21.

Spiessens, P. y Manderick, B., «A massively parallel genetic algorithm: Implementation and first analysis», en Belew, R.K. y Booker, L.B. (Eds.), *Proceedings of the Fourth International Joint Conf. On Genetic Algorithms*, Morgan Kaufmann Publishers, (1991), 279-287.

Stork, D.G. y Allen, J.D., «How to solve the N-bits parity problem with two hidden units», *Neural Networks*, Vol. 5, (1992), 923-926.

Sunderam, V.S., «PVM: A framework for parallel distributed computing», *Concurrency, Practice and Experience*, Vol. 2, N. 4, (1990), 315-339.

Tanese, R., «Parallel Genetic Algorithms for a Hypercube», en Grefenstette, J.J. (Ed.), *Proceedings of the Second International Conference on Genetic Algorithms*, Lawrence Erlbaum Associates, 1987, 177-184.

Widrow, B. y Hoff, M.E., «Adaptive switching circuits», *1960 IRE WESTCON Convention Record*, (1960), 96-104.

Williams, R.J. y Zipser, D., «A learning algorithm for continually running fully recurrent neural networks», *Neural Computation*, Vol. 1, (1989), 270-280.

Whitley, D. y Starkweather, T., «Optimizing Small Neural Networks Using a Distributed Genetic Algorithm», *Proceeding of the International Joint Conf. Neural Networks*, Lawrence Erlbaum, (1990), 206-209.

Whitley, D., Starkweather, T. y Bogart, C., «Genetic Algorithms and Neural Networks: Optimizing Connections and Connectivity», *Parallel Computing*, Vol. 14, (1990), 347-361.

Yao, X., «A review of evolutionary artificial neural networks», *Technical Report*, Commonwealth Scientific and Industrial Research Organisation, PO Box 56, Highett, Victoria 3190, Australia, (1992).

Yao, X., «An empirical study of genetic operators in genetic algorithms», *Microprocessing and Microprogramming*, Vol. 38, (1993), 707-714.