Rev. Int. Mét. Num. Cálc. Dis. Ing.
Vol. **18, 4,** 573-593 (2002)

Revista Internacional de Métodos Numéricos para Cálculo y Diseño en Ingeniería

Optimización basada en confiabilidad por medio de redes neuronales y algoritmos evolutivos

Jorge Hurtado y Diego Álvarez
Universidad Nacional de Colombia
Apartado 127
Manizales, Colombia
Tel.: 57-68-863 182, Fax.: 57-68-863 906
e-mail: jhurtado@emtelsa.multi.net.co

Resumen

A diferencia de la optimización usual, la optimización estocástica consiste en una minimización de una función de coste sujeta a condiciones expresadas en forma de probabilidades en lugar de funciones deterministas, lo que la convierte en un problema mucho más complejo. En esta clase de optimización son comunes los problemas donde la función o funciones objetivo y sus condiciones son el producto de un algoritmo numérico bastante complicado que no es ni diferenciable ni explícito. En estos casos no es posible utilizar los algoritmos de optimización basados en el gradiente y, además, los tiempos de evaluación funcional para el cálculo de las probabilidades hacen el problema inabordable. En el presente artículo se propone y se evalúa un procedimiento para superar estas dificultades utilizando redes neuronales artificiales y algoritmos evolutivos.

Palabras clave:
optimización estocástica, algoritmos evolutivos, redes neuronales artificiales, confiabilidad estructural.

RELIABILITY-BASED OPTIMIZATION USING NEURAL NETWORKS AND EVOLUTIONARY ALGORITHMS

Summary

At a difference to conventional optimization, reliability-based optimization aims at the minimization of a cost function subject to conditions formulated in terms of probabilities instead of the usual deterministic functions. This renders it a more involved problem. In addtition, it is often the case that the primal functions and the conditions are the output of an involved numerical algorithm which is both non-differentiable nor explicit. In these cases it is not possible to use the gradient of the function. Besides, the high computational times for calculating the probabilities make the problem untractable. In this paper an approach that merges neural networks and evolutionary algorithms is proposed for overcoming these difficulties. The proposed procedure allows finding an optimal solution with stochastic conditions in a short computation time.

Keywords:
stochastic optimization, evolutionary algorithms, neural networks, structural reliability.

 ISSN: 0213–1315 Recibido: Julio 2001

INTRODUCCIÓN

El diseño de sistemas estructurales complejos busca que se satisfagan los criterios de seguridad, serviciabilidad, durabilidad, economía y, en ocasiones, belleza; de este modo, el diseño de los sistemas está regulado por normas y códigos que imponen métodos y condiciones que hacen de esta una complicada y dispendiosa tarea: encontrar el mejor diseño posible sin comprometer la confiabilidad del mismo (*diseño robusto*). Los parámetros por optimizar son las variables de diseño. Con el fin de comparar las diferentes alternativas viables de diseño es necesario formular un objetivo, siendo el costo del prototipo el más usual. En la optimización se busca minimizar este objetivo, cumpliendo, asimismo, con algunas condiciones, fijadas por el criterio del diseñador o por las normas que lo gobiernen, como podrían ser las dimensiones mínimas o máximas de los elementos, las tensiones permisibles en éstos, la resistencia de los materiales, etc.

La optimización estructural se realiza usualmente de modo determinista. Esta optimización consiste en minimizar la función de coste $J(\vec{X})$ sujeto a condiciones deterministas de desigualdad. Simbólicamente esto se expresa como

$$\begin{aligned} \text{minimizar} \quad & J(\vec{X}) \\ \text{sujeto a} \quad & f_1(\vec{X}) < F_1 \\ & f_2(\vec{X}) < F_2 \\ & \vdots \end{aligned} \tag{1}$$

En esta ecuación $\vec{X}$ es el vector de variables de diseño, las $f_i(\vec{X})$ son respuestas del sistema que dependen de estas variables y F_i son condiciones impuestas a estas respuestas. La mayor desventaja de este enfoque es que las incertidumbres inherentes al sistema no son tenidas en cuenta.[1] Si modelamos estas incertidumbres haciendo uso de la Teoría de Probabilidades, esta optimización se convierte en[2]

$$\begin{aligned} \text{minimizar} \quad & J(\vec{X}) \\ \text{sujeto a} \quad & \mathrm{P}[f_1(\vec{X}) > F_1] \leq P_1 \\ & \mathrm{P}[f_2(\vec{X}) > F_2] \leq P_2 \\ & \vdots \end{aligned} \tag{2}$$

donde $\mathrm{P}[A]$ es la probabilidad del evento aleatorio A y las P_i son máximos valores admisibles para las probabilidades de fallo. Esta optimización estocástica de sistemas involucra, en consecuencia, los siguientes pasos:

1. Definición probabilista de las variables que intervienen en el sistema.
2. Representación del comportamiento del sistema mediante el uso de un modelo matemático adecuado que incluya a su vez la evaluación de las condiciones al modelo.
3. Encontrar la mejor configuración para este sistema de tal modo que se minimicen las funciones objetivo que representan los criterios a optimizar, respetando las condiciones impuestas.

En este trabajo se utilizará como sistema complejo de referencia una estructura metálica. Sin embargo, mucho de lo que se enuncie aquí es válido para cualquier otro tipo de sistema

complejo que se pretenda optimizar en condiciones de incertidumbre de los parámetros que lo gobiernen.

La optimización estocástica de estructuras es, en la mayoría de los casos, difícil de realizar, ya que la representación matemática del sistema en puede requerir altos tiempos de evaluación computacional. Esto se debe a que su modelación requiere el cálculo de la probabilidad de fallo del sistema, para lo cual se recurre comúnmente a técnicas igualmente costosas como la simulación de Monte Carlo, debido a que las funciones de comportamiento límite no son explícitas. Por otro lado, como en ocasiones las funciones de coste y de condiciones tampoco son explícitas, ni continuas ni diferenciables, es necesario adoptar un método de optimización capaz de enfrentar tales casos, ya que los métodos clásicos de optimización matemática basados en el uso del gradiente, como la técnica del descenso más empinado, los métodos de Newton, quasi-Newton o los métodos del gradiente conjugado, son ineficientes para realizar la optimización global en tales situaciones[3]. Todo esto hace que la optimización basada en confiabilidad, que ha sido definida como *la meta última del diseño estructural*[1] se presente como un objetivo mucho más costoso en términos computacionales que la optimización clasica o el análisis probabilista por simulación de Monte Carlo, considerados separadamente.

Con el fin de simplificar este problema, se ha desarrollado en este artículo un enfoque que utiliza redes neuronales artificiales (RNA) para representar el comportamiento del sistema (Paso 2) empleándolas como un substituto en la evaluación del modelo numérico, con el objeto de reducir los tiempos de simulación conservando la precisión y generalización del modelo inicial. Esto se combina con algoritmos evolutivos (AE) como método de optimización global de las funciones de coste, teniendo en cuenta las condiciones probabilistas (Paso 3), debido a su versatilidad para trabajar con funciones implícitas, de manera que no se hace necesario conocer ni estimar su gradiente.

A continuación se describirán la representación de sistemas con RNA y la optimización con los AE empleados. Luego se utilizarán los AE conjuntamente con las RNA para optimizar el diseño y las variables del proceso, se expondrán los modelos matemáticos del ejemplo por resolver y finalmente se discutirán los resultados obtenidos y el método empleado. En el artículo se incluye un apéndice en el que se resumen las ecuaciones básicas sobre las redes neuronales y los algoritmos evolutivos usados en esta investigación.

REPRESENTACIÓN DE SISTEMAS CON RNA

Las RNA se han utilizado ampliamente en tareas tales como clasificación, aproximación funcional, optimización, reconocimiento de patrones y compresión de datos en diversos campos de la ingeniería y la estadística.[3,4] Las redes están formadas por capas de elementos de procesamiento o neuronas que se conectan entre sí. Según su arquitectura pueden usarse para diferentes propósitos. Entre éstos se encuentra la aproximación de cualquier función no lineal, sea esta continua o discontinua, por medio de la técnica conocida como aprendizaje supervisado. Esta labor consiste en calcular los parámetros de la red de acuerdo a algunos casos ejemplares de los cuales se conocen las respuestas del sistema a datos de entrada. De esta suerte, la red ya entrenada puede ser utilizada para estimar las respuestas correspondientes a nuevos datos de entrada. Esto las hace ideales para sustituir el programa de cálculo de estructuras por elementos finitos u otro método numérico, requerido repetidamente por la simulación intensa de Monte Carlo, ya que ésta consiste justamente en calcular las respuestas debidas a entradas aleatorias. Por esta razón, el enfoque del aprendizaje supervisado es el adoptado en lo que sigue.

Los siguientes pasos están involucrados en el uso de RNA para aproximación funcional en tal tipo de aprendizaje:

1. Selección del modelo de RNA a utilizar. Existen varias arquitecturas de RNA que pueden utilizarse como aproximadores funcionales; entre ellas se encuentran las RNA Multicapa de Alimentación Hacia Adelante (MCAHA) y las de Funciones de Base Radial (BR); cada una de ellas tiene definidos sus propios parámetros y estructuras y están ampliamente descritas en la bibliografía del tema[3,4].

2. Determinación de las entradas y salidas necesarias de la RNA. Las RNA se deben utilizarse en principio para aproximar las funciones que sean de lenta evaluación computacional. Para ello se deben identificar aquellas funciones y condiciones de difícil evaluación y las variables más directamente relacionadas con ellas, de tal forma que sean respectivamente las salidas y las entradas de la RNA.

3. Cálculo del conjunto de datos de entrenamiento/validación. El éxito de la aproximación funcional hecha con la RNA depende principalmente de la calidad de los datos de entrenamiento. Cuando se utilizan RNA, uno de los principales problemas es determinar cuántos pares de conjuntos de entrenamiento son necesarios para aprender a simular correctamente el modelo numérico propuesto. Se requiere seleccionar el menor conjunto de entrenamiento que permita la buena generalización del sistema. Como las RNA tienen capacidades de interpolación mas no de extrapolación, los datos usados para el entrenamiento deben cubrir todo el dominio de simulación de modo que se evite cualquier extrapolación. Por otra parte, se debe tener presente que en las últimas generaciones de los AE es común obtener puntos solución cercanos a la región de puntos inviables, razón por la cual cualquier error en la identificación de esta región conlleva a una mala respuesta.

   Por todas estas razones los puntos del conjunto de entrenamiento, en el ejemplo numérico que consta más adelante, se determinaron con una técnica de muestreo llamada Muestreo del Hipercubo Latino Mejorado (MHLM)[5,6], porque este método tiene la propiedad de generar económicamente muestras aleatorias en todos los rangos de valores posibles de las mismas, conservando la Función de Distribución de Probabilidades (FDP) marginal de cada variable simulada. Además el MHLM produce estimadores casi insesgados de los más importantes parámetros estadísticos (media, momentos, etc.) y su varianza es reducida en comparación con la técnica clásica de Monte Carlo simple. Para este propósito se ha seguido la recomendación de que el conjunto de entrenamiento tenga un tamaño al menos de cinco veces el número de parámetros de la red, ya que los datos están libres de ruido, pues no se obtienen a partir de experimentos sino de manera puramente numérica[3].

   Por otro lado se necesita un conjunto de validación de modo tal que se pueda evaluar la eficiencia de la aproximación dada por la RNA entrenada; este conjunto debe ser generado de la misma forma que el conjunto de entrenamiento.

4. Entrenamiento. El uso de la RNA involucra dos diferentes tareas: la definición de la RNA (estructura, número de neuronas, funciones de transferencia, etc.) y de estimación de sus parámetros.

   La estructura de la RNA gobierna la capacidad de la misma para proveer una aproximación adecuada de las relaciones de entrada y salida. Esta estructura se determina por ensayo y error hasta encontrar una RNA con la configuración adecuada.

5. Fase de Aprendizaje. Se deben entrenar varias RNA porque el espacio de búsqueda con los pesos contiene varios mínimos locales. El variar las condiciones iniciales en el entrenamiento de la RNA puede llevar a diferentes soluciones. Para cada función que se aprende, se utiliza la mejor solución de varios entrenamientos.

Después de esto, se espera obtener una RNA con buenas propiedades de generalización, aproximación de la función objetivo y/o condiciones iniciales. Aunque algunos autores recomiendan entrenar una RNA para cada función a aproximar, queda a criterio del programador si se utiliza una sola RNA o varias RNA para simular las funciones deseadas[3].

6. Validación de las RNA entrenadas. Esta fase consiste en verificar la estimación de la función con la red utilizando datos no presentados a ella durante el entrenamiento. Para el análisis que nos ocupa la generación de un conjunto de validación implica llamadas adicionales del programa de cálculo exacto de la función. En general se trata de una fase opcional, de la cual puede prescindirse si se tiene seguridad sobre la aproximación lograda por experiencia previa.

Los tiempos de entrenamiento y los tiempos de simulación con la RNA varían con la arquitectura de la misma y el algoritmo de entrenamiento usado. Para que la implementación sea eficiente se debe tener en cuenta que los tiempos de entrenamiento y de simulación con la RNA deben ser mucho menores que el tiempo utilizado para calcular la respuesta con el modelo matemático nominalmente exacto.

OPTIMIZACIÓN CON ALGORITMOS EVOLUTIVOS

Los algoritmos evolutivos (AE) son algoritmos que transforman poblaciones de objetos numéricos individuales en nuevas poblaciones usando operaciones inspiradas en la genética y la evolución de las especies (principio de la selección natural), tales como la regla darwiniana de sobrevivencia y reproducción de los más fuertes. En los años recientes han encontrado aplicación en múltiples áreas de la técnica y la ciencia[2].

Los AE son un método de optimización muy robusto, eficiente y convergente. A dos principales características se les atribuye este hecho[2]:

- Los AE tienen un patrón distintivo de búsqueda en el espacio objetivo: en vez de una búsqueda punto a punto, el conjunto de búsqueda se expande a un grupo en el cual un punto de búsqueda se mueve entre picos; por tanto, se incrementa la probabilidad de encontrar una solución cercana a la posición del óptimo global.

- El uso de funciones objetivo y de condiciones no requiere el uso de gradientes u otro conocimiento auxiliar. En consecuencia, la búsqueda con AE no está restringida a condiciones de continuidad, no linealidad o convexidad de la función objetivo o las condiciones.

Los siguientes son los pasos de diseño con AE:

1. Selección del AE y ajuste de sus parámetros. Existen dos tipos principales de AE: las estrategias evolutivas (EE)[7] y los algoritmos genéticos (AG)[2]. De cada uno de ellos existen variantes y fusiones ideadas para mejorar su desempeño en problemas de optimización uni- y multiobjetivo con y sin condiciones.

   Para la optimización basada en confiabilidad que se estudia aquí se deben manejar condiciones de tipo probabilista, como ha quedado dicho anteriormente. En general, un AE que no esté pensado para tratar problemas condicionados se puede modificar mediante el uso de funciones de penalización, ampliando la función de aptitud del algoritmo. Sin embargo, el uso de estas funciones de penalización tiene algunas limitaciones importantes, siendo la principal dificultad su selección adecuada, ya que son normalmente generadas por ensayo y error, de modo que su mala definición puede afectar la convergencia de los resultados[8]. Para el caso que nos ocupa se debe por

tanto utilizar AE que puedan realizar la minimización con condiciones y evitar así el uso de funciones de penalización.

2. Determinación de los valores de las variables de diseño. Una vez definido el problema de optimización (variables de diseño, funciones objetivo a minimizar y condiciones), se procede a utilizar el AE seleccionado. Como primera medida, las variables de diseño adoptan unos valores iniciales (que pueden ser a priori o seleccionados aleatoriamente) que son codificados según las necesidades del algoritmo (codificación real o codificación binaria). Las variables de diseño son optimizadas simultáneamente para que todas las configuraciones encontradas sean idóneas, según el problema. Una vez codificados los individuos, a cada uno se le evalúa su aptitud (la cual representa la adaptabilidad del individuo al medio), calculada con base en la función objetivo y en la medida de violación de las condiciones. En el método propuesto se emplea la RNA seleccionada como sustituto del modelo original para el cálculo de las condiciones probabilistas.

   Cada individuo es seleccionado según su aptitud en el medio de acuerdo al algoritmo evolutivo utilizado. Se supone que los individuos con una aptitud alta tienen mayores posibilidades de reproducirse y pasar sus características a la siguiente generación. Así pues, se forma un banco de apareamiento, que es una población intermedia que será sujeta a combinación. De este banco se seleccionan aleatoriamente individuos que intercambian sus codificaciones según un patrón de combinación definido de tal modo que se generan nuevas descendencias que posteriormente son sujetas a mutación. Los nuevos individuos son sujetos a mutación de algunos genes con el objeto de evitar la pérdida potencial de material genético de los procesos de selección y combinación, para prevenir el llamado desvío genético.

   Una variante de los AE que mejora significativamente el desempeño del algoritmo es el elitismo, que consiste en que el mejor individuo entre padres e hijos pasa a la siguiente generación intacto, sin ser alterado por la mutación[9].

   Una vez que se ha formado la nueva generación, ésta se convierte en la nueva población de padres, a los cuales se les evaluará su aptitud y serán sometidos a las operaciones de selección, combinación y mutación nuevamente. Esto se hará hasta que se cumpla algún criterio para finalizar la ejecución del algoritmo, por ejemplo, que se alcance el número máximo de generaciones o que después de algún número definido de generaciones no se haya alcanzado un nuevo éxito.

En este trabajo se propone, entonces, una simbiosis de los algoritmos evolutivos como técnica de minimización de la función de coste y las redes neuronales como método de cálculo de las probabilidades de fallo que constituyen las restricciones de la minimización.

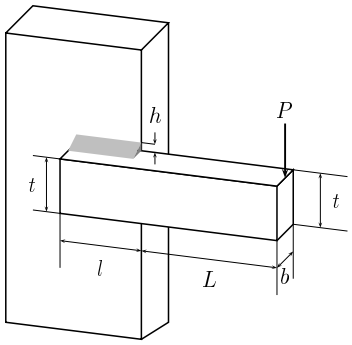

Figura 1. Esquema de la viga en voladizo

CASO DE ESTUDIO

El enfoque anteriormente resumido se ha aplicado a la optimización del costo de una viga en voladizo soldada. La formulación de este caso se tomó de,[10] pero para efectos de estudiar la optimización con condiciones probabilistas se modificaron las fórmulas (4), (5), (9) y (10), que aparecen a continuación. Éstas conciernen a varias condiciones probabilistas sobre la tensión cortante (τ), la tensión por flexión en la viga (σ), la carga de pandeo en la barra (P_c), la deflexión final en la viga (δ) y a las dimensiones. Existen cuatro variables de diseño como se muestra en la Figura 1: $h(x_1)$, $l(x_2)$, $t(x_3)$, y $b(x_4)$.

Nuestro problema es: minimizar

$$f(\vec{X}) = 1,10471{x_1}^2 x_2 + 0,04811 x_3 x_4 (14,0 + x_2) \tag{3}$$

sujeto a las siguientes condiciones

$$g_1(\vec{X}) = \mathrm{P}\left[\tau(\vec{X}) > \tau_{\max}\right] \leq 0,001 \tag{4}$$

$$g_2(\vec{X}) = \mathrm{P}\left[\sigma(\vec{X}) > \sigma_{\max}\right] \leq 0,001 \tag{5}$$

$$g_3(\vec{X}) = x_1 - x_4 \leq 0 \tag{6}$$

$$g_4(\vec{X}) = 0,10471 x_1^2 + \\ + 0,04811 x_3 x_4 (14,0 + x_2) - 5,0 \leq 0 \tag{7}$$

$$g_5(\vec{X}) = 0,125 - x_1 \leq 0 \tag{8}$$

$$g_6(\vec{X}) = \mathrm{P}\left[\delta(\vec{X}) > \delta_{\max}\right] \leq 0,001 \tag{9}$$

$$g_7(\vec{X}) = \mathrm{P}\left[P > P_c(\vec{X})\right] \leq 0,001 \tag{10}$$

donde,

$$G = \frac{E}{2(1-\nu)} \tag{11}$$

$$R = \sqrt{\frac{x_2^2}{4} + \left(\frac{x_1 + x_3}{2}\right)^2} \tag{12}$$

$$M = P(L + \tfrac{x_2}{2}) \tag{13}$$

$$J = 2\left\{\sqrt{2} x_1 x_2 \left[\frac{x_2^2}{12} + (\tfrac{x_1 + x_3}{2})^2\right]\right\} \tag{14}$$

$$\tau' = \frac{P}{\sqrt{2} x_1 x_2} \tag{15}$$

$$\tau'' = \frac{MR}{J} \tag{16}$$

$$\tau(\vec{X}) = \sqrt{(\tau')^2 + 2\tau'\tau''\frac{x_2}{2R} + (\tau'')^2} \tag{17}$$

$$\sigma(\vec{X}) = \frac{6PL}{x_4 x_3^2} \tag{18}$$

$$\delta(\vec{X}) = \frac{4PL^3}{E x_3^3 x_4} \tag{19}$$

$$P_c(\vec{X}) = \frac{4{,}013E\sqrt{\frac{x_3^2 x_4^6}{36}}}{L^2}\left(1 - \frac{x_3}{2L}\sqrt{\frac{E}{4G}}\right) \tag{20}$$

La carga P y el módulo de elasticidad en la viga E se definen como variables aleatorias independientes distribuídas según leyes normal y lognormal respectivamente, con valores de momentos $\mu_P = 6000$ lb, $\sigma_P = 600$ lb, $\mu_E = 30 \times 10^6$ psi y $\sigma_E = 30 \times 10^5$ psi. En

estas definiciones $\mu_{\cdot}$ representa la media y $\sigma_{\cdot}$ la desviación estándar de la variable implicada. Además se dan los siguientes valores: $L = 14$ in, $\nu = 0{,}25$, $\tau_{\max} = 13600$ psi, $\sigma_{\max} = 30000$ psi y $\delta_{\max} = 0{,}25$ in. Los rangos de búsqueda de las variables aleatorias de diseño están dados por: $0,1 \leq x_1 \leq 2,0$; $0,1 \leq x_2 \leq 10,0$; $0,1 \leq x_3 \leq 10,0$ y $0,1 \leq x_4 \leq 2,0$.

Las ecuaciones (4), (5), (9) y (10), $\vec{g} = (g_1(\vec{X}), g_2(\vec{X}), g_6(\vec{X}), g_7(\vec{X}))$, son las condiciones probabilistas. En el entrenamiento de la RNA usada como sustituto del cálculo de la viga para calcularlas se ha empleado, como conjunto de entrada, el valor de las variables de diseño $\vec{X} = (x_1, x_2, x_3, x_4)$ generado previamente con MHLM, utilizando para cada una de las variables densidades de probabilidad uniformes sobre la región de búsqueda, y como conjunto de salida el valor de las funciones $\vec{g}$. Se debe tener en cuenta que en el conjunto de entrenamiento no aparecen explícitos los valores de P o de E, pero sí se encuentran implícitamente en $\vec{g}$. En total se generaron 600 muestras para el conjunto de entrenamiento y 1000 para el conjunto de validación.

En caso de que la función objetivo dependa de las variables aleatorias, se podría calcular la aptitud del individuo como la media o valor esperado de las aptitudes para cada una de sus simulaciones de Monte Carlo[11].

Se utilizaron unas RNA MCAHA con una capa de entrada (para $\vec{X}$), dos ocultas y una de salida, para aproximar las condiciones $\vec{g}$, ya que, como se comenta en la referencia 3, en ocasiones una capa oculta no es suficiente para tratar con discontinuidades en las funciones que se quiere aproximar. Se utilizaron neuronas con funciones de transferencia sigmoides bipolares en las capas ocultas y lineales en la capa de salida y además umbrales para cada neurona.

Para evitar que se saturaran las funciones sigmoides, se normalizaron las entradas y las salidas de las RNA según

$$\bar{x}_i = \frac{x_i - \mu_i}{\sigma_i} \tag{21}$$

donde x_i es el vector con el cual se entrena o valida la RNA MCAHA y μ_i y σ_i son vectores de la media y la desviación estándar de variable x_i. Como función de costo se empleó una basada en el error medio cuadrático, definido como

$$EMC = \frac{1}{NM} \sum_{j=1}^{N} \sum_{k=i}^{M} (y_{jk} - \hat{y}_{jk})^2 \tag{22}$$

donde N es el tamaño del conjunto de entrenamiento, M es el número de neuronas en la capa de salida y $\hat{y}_{jk}$ es la estimación hecha por la RNA de y_{jk}, que es la salida correcta. Para entrenar las RNA se utilizó el algoritmo Gauss–Newton–Levenberg–Marquardt (GN-LM) porque alcanza en igual número de épocas de entrenamiento menor error en la función de costo de la RNA que los algoritmos clásicos de retropropagación, quasi–Newton o gradiente conjugado, aunque GN-LM requiere más memoria que éstos. Para prevenir el fenómeno de sobreajuste se utilizó la técnica de la finalización temprana, la cual sigue el desempeño de la RNA durante la fase de entrenamiento, que termina cuando el error en el conjunto de validación alcanza un mínimo[12]. Se ensayaron varias configuraciones de RNA modificando la cantidad de neuronas en las capas ocultas y los pesos iniciales, obteniendo finalmente una RNA adecuada con siete y cuatro neuronas en sus capas ocultas. En el Apéndice se encuentra una descripción somera de esta red.

Una vez entrenada la RNA se utilizó como substituto del valor de $\vec{g}$, evaluando las demás condiciones y la función objetivo directamente con sus fórmulas.

Como AE se utilizó una EE especificada en la referencia 13 llamada EVOSLINOC combinada con la EE descrita en la referencias 14 y 7, ya que el problema es uniobjetivo sujeto a varias condiciones. La selección de este último algoritmo se hizo con base en

que utiliza el valor de las violaciones a las condiciones para buscar la región de soluciones viables, fomentando la diversidad en las soluciones y haciendo que la población explore en el espacio de puntos factibles sin utilizar funciones de penalización. Además este algoritmo propende la incentivación de la diversidad de soluciones a través de la creación de nichos. Este algoritmo también se resume en el Apéndice.

Se ejecutó el algoritmo EVOSLINOC 30 veces, con los siguientes parámetros (definidos en el Apéndice):

- $(\mu + \lambda) = (10 + 50)$, número de padres y de hijos respectivamente, con selección de los mejores individuos entre padres e hijos para la siguiente generación.
- $\beta = 0,00001$, parámetro de definición de las aptitudes $\mathcal{N}_f$, $\mathcal{N}_i$, $\mathcal{N}_{if}$.
- $p = 5$, parámetro de la distancia de Minkowski para la definición del espacio de condiciones $\mathcal{C}$.
- $\mathcal{C}extra = 0,05$, parámetro de definición de la clase adicional.

Conjuntamente se utilizó como EE el algoritmo descrito en la referencia 15, resumido en el Apéndice, con los siguientes parámetros de mutación:

- $\varepsilon_\sigma = 0,001$
- $\tau_0 = 0,5$
- $\tau = 0,5$
- $\beta = 0,0873$

Como criterios de finalización establecidos se adoptaron los siguientes: no encontrar un mejor resultado en 1000 generaciones o bien alcanzar el número máximo de generaciones (2000).

Estadístico	10 individuos	1600 individuos
Mejor solución	2,34142	2,47786
Peor solución	3,32583	2,96011
Media	2,67741	2,68982
Desviación estándar	0,23188	0,12169

Tabla I. Resultados encontrados variando la población inicial para el diseño óptimo de una viga soldada

La población inicial para el AE se puede seleccionar a partir del conjunto de puntos de entrenamiento y validación aprovechando el hecho que son muchos puntos (1600) uniformemente distribuídos sobre el espacio de búsqueda de los que se les conoce exactamente sus condiciones de más difícil evaluación. Para generar la población inicial del AE se utilizaron dos variantes: utilizar como padres 10 puntos generados aleatoriamente en la región de búsqueda y emplear como población inicial todos los puntos con los que se entrenaron y validaron las RNA (1600 puntos). Se observó que utilizar este último método conlleva a obtener resultados parecidos, ya que al tener una población inicial tan grande distribuída sobre la región de búsqueda, los resultados se focalizan sobre la misma región; por otro lado, se utilizó el AE pero con el primer enfoque, encontrándose mejores soluciones para las mismas 30 corridas. Un resumen de los estadísticos de las soluciones encontradas se presenta en la Tabla I.

Se puede observar que el enfoque con 1600 padres como población inicial tiene una desviación estándar y una peor solución menor; sin embargo, el enfoque con 10 individuos como población inicial, a pesar de tener soluciones mucho más dispersas, encuentra mejores soluciones. Por lo tanto no se recomienda utilizar como población inicial todos los datos de entrenamiento y validación de la RNA, aunque la población inicial de 10 padres si se podría escoger del conjunto de entrenamiento.

El AE prosigue y cuando se encuentra un punto donde la población se detiene por más de 50 generaciones, se evalúan de nuevo las condiciones y la función objetivo, pero en vez de utilizar la RNA, se evalúan con las funciones originales, buscando con ello generar unos nuevos puntos de entrenamiento para la RNA que actualmente está en uso. Se unen estos puntos con el anterior conjunto de entrenamiento y con este se entrena de nuevo la RNA: esto permite ajustar la superficie de respuesta generada por la RNA a la región de búsqueda actual. Por lo general, algunos puntos que la RNA consideraba como viables ahora no lo son y en consecuencia el algoritmo evolutivo tiene que ajustar las aptitudes de sus individuos, hecho que se ve reflejado en la historia de la evolución, como se puede observar en la Figura 2 (en las generaciones 351, 479, 574 y 1160). Se continúa la ejecución del algoritmo evolutivo hasta que se cumplen los criterios de finalización impuestos. El AE frecuentemente encuentra mejores puntos y cuando se vuelve a estancar en una solución, se utiliza de nuevo la técnica propuesta para corregir la superficie de respuesta dada por la RNA. Como es común ejecutar varias veces el AE sobre el problema para encontrar diferentes soluciones óptimas, los puntos que se van agregando al conjunto de entrenamiento de la RNA pueden ir acumulándose de ejecución en ejecución, esperando así mejorar progresivamente el ajuste de la RNA a las funciones verdaderas que se quieren imitar.

Finalmente, al terminar el AE, se evalúan las condiciones del conjunto de padres así como las funciones objetivo para establecer si son o no viables las soluciones encontradas y de este conjunto el diseñador puede seleccionar el diseño óptimo deseado.

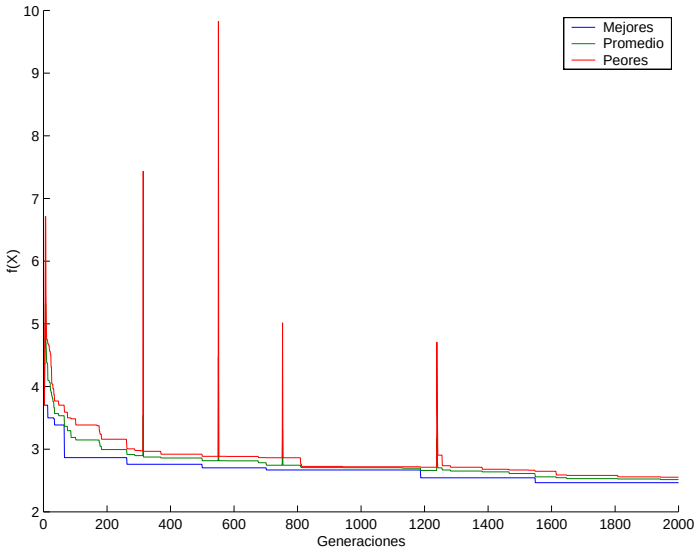

Figura 2. Historia de la evolución del AE utilizado, para la mejor optimización realizada que aparece en la Tabla II

Los mejores resultados encontrados aparecen en la Tabla II. Se podrían encontrar mejores soluciones si se dejase correr el algoritmo con mayor número de generaciones empezando con poblaciones iniciales diferentes.

En promedio para cada una de las veces que se utilizó el AE se necesitó evaluar 50 veces las funciones $g(\vec{X})$ originales (sin incluir las 600 veces que se evaluó el conjunto de entrenamiento y las 1000 que se hizo con el de validación y que éstos sólo se evaluaron un vez) y 99,750 veces la RNA, con lo que se puede demostrar un ahorro significativo en el número de evaluaciones que se deben hacer a las funciones originales.

En cuanto a los tiempos de evaluación, éstos se reducen de 273,9825 segundos para evaluar las condiciones $g(\vec{X})$ de los 50 hijos de una generación utilizando las funciones originales a 0,2523 segundos para evaluar los mismos individuos con la RNA, mostrando una disminución en aproximadamente 1086 veces en el tiempo de evaluación de las funciones $g(\vec{X})$.

Variable de diseño	Algoritmo EVOSLINOC
$x_1(h)$	0,23607178216068
$x_2(l)$	3,91062776445514
$x_3(t)$	9,83014452933563
$x_4(b)$	0,24800003000565
$g_1(\vec{X})$	0,00015
$g_2(\vec{X})$	0,00002
$g_3(\vec{X})$	-0,01192824784497
$g_4(\vec{X})$	-2,89349467718363
$g_5(\vec{X})$	-0,11107178216068
$g_6(\vec{X})$	0
$g_7(\vec{X})$	0
$F(\vec{X})$	2,34142906324074

Tabla II. Mejores resultados encontrados para el diseño óptimo de una viga soldada

Un esquema del algoritmo del método propuesto se encuentra en la Figura 3.

ANÁLISIS DE RESULTADOS

Existen algunos puntos que se deben someter a discusión sobre el enfoque propuesto:

Generación del conjunto de entrenamiento

En casos en que sea difícil obtener los datos para el conjunto de entrenamiento/validación podrían utilizarse técnicas como el llamado *jittering*, ya que a medida que se ejecuta el algoritmo evolutivo se mejora progresivamente el conjunto de entrenamiento en las regiones donde es prioritario.

Desempeño de la RNA

El principal problema observado es el ajuste de la superficie de respuesta generada por la RNA a los puntos de entrenamiento, principalmente a aquellos que se agregan cada vez que la búsqueda se estanca, ya que generalmente estos puntos se encuentran muy cerca de la frontera que delimita los puntos viables.

```
Seleccionar el modelo de RNA por utilizar; Determinar las entradas
y salidas necesarias de la RNA; Calcular el conjunto de datos de
entrenamiento/validaci\'on; Entrenar y validar la RNA; Seleccionar
el AE y ajustar sus par\'ametros; PADRES = Seleccionar de la
poblaci\'on inicial a partir del
    conjunto de entrenamiento/validaci\'on;
Calcular aptitudes de PADRES;

Ngen = 0;          %Contador del n\'umero de generaciones
LWS = 0;           %Contador de generaciones sin \'exito
MAXgen = 2000;     %M\'aximo n\'umero de generaciones
MAXLWS = 1000;     %M\'aximo n\'umero de generaciones sin \'exito

MIENTRAS (Ngen < MAXgen) Y (LWS < MAXLWS)
  Ngen = Ngen + 1;
  HIJOS = Mutaci\'on(Combinaci\'on(PADRES));
  Calcular las funciones objetivo/condiciones de HIJOS
      utilizando la RNA y el modelo  matem\'atico original
      seg\'un corresponda;
  Calcular aptitudes de HIJOS;
  PADRES_OLD = PADRES;
  PADRES = Seleccionar mejores individuos de (PADRES+HIJOS);

  SI (mejor individuo PADRES) es mejor que
                                (mejor individuo PADRES_OLD)
    LWS = 0;
  DE_LO_CONTRARIO
    LWS = LWS + 1;
  FIN_SI

  SI LWS = 50
    Calcular las funciones objetivo/condiciones de PADRES
      utilizando el modelo matem\'atico original;
    Calcular las aptitudes de PADRES;
    Agregar PADRES al conjunto de entrenamiento de la RNA;
    Entrenar la RNA;
  FIN_SI
FIN_MIENTRAS Calcular las funciones objetivo/condiciones de PADRES
  utilizando el modelo matem\'atico original;
Verificar si se cumplen las condiciones; Seleccionar el individuo
\'optimo de PADRES; FIN
```

Figura 3. Esquema del algoritmo propuesto

Uso del algoritmo evolutivo

La más grande dificultad encontrada radica en ajustar apropiadamente los parámetros del AE para que este funcione óptimamente, porque estos parámetros se ajustan experimentalmente. Este problema, sin embargo, no es concerniente a la optimización en sí, sino al AE utilizado.

Ahorro en número de operaciones y disminución de los tiempos de simulación

El propósito fundamental de la aproximación funcional utilizando las RNA es evitar al máximo evaluar el modelo matemático que describe el sistema; como consecuencia, se reducen los tiempos de simulación. Como se pudo observar, en el caso de estudio, un ahorro del 1995 veces en evaluaciones funcionales permite una disminución de 1086 veces en el tiempo de simulación para el problema analizado. Es muy probable que en problemas más complicados se tenga un ahorro de tiempo mayor.

CONCLUSIONES

El objetivo principal de este artículo ha sido el de proponer a modo de ejemplo una forma para resolver tal complejidad. Se presentó un enfoque que involucraba la representación del modelo matemático del comportamiento de la estructura con una RNA y una optimización estocástica realizada por un AE para la optimización del diseño de una viga.

La optimización basada en confiabilidad es un problema complejo con muchas soluciones potenciales y varias funciones objetivo que no son ni continuas ni diferenciables ni explícitas.

La aplicación del AE con condiciones probabilistas que usa la RNA como aproximador funcional es particularmente útil porque de otro modo el problema sería dificil de abordar con técnicas usuales de optimización estocástica, debido a la no linealidad de las funciones de restricción y de comportamiento límite y a su naturaleza implícita. El uso de la RNA reduce notablemente el tiempo de cálculo comparados con los diferentes modelos que utilizan los métodos numéricos clásicos, lo cual permite calcular fácilmente la probabilidad de fallo en cada caso. Esto es debido a su habilidad para aproximar funciones. En el ejemplo considerado en este trabajo, el algoritmo evolutivo optimización conocido como EVOSLINOC resultó adecuado para esta clase de problema. Se obtuvieron varias soluciones excelentes de un modo relativamente rápido.

Este enfoque reduce drásticamente los tiempos de simulación comparados con el enfoque clásico, mientras mantiene buena precisión y propiedades de generalización. Usando una representación con AE y RNA una sola corrida del AE que duraría más de 152 horas se puede simular en menos de 8,5 minutos con un PC mediante el uso de la RNA.

AGRADECIMIENTOS

Para la realización de esta investigación se contó con apoyo económico de la Universidad Nacional de Colombia, entidad a la cual los autores expresan su agradecimiento.

REFERENCIAS

1 J-S.R. Jang, C.T. Sun y E. Mizutani, "*Neuro fuzzy and soft computing*", Prentice Hall, EE.UU., (1997).

2 G.C.S. Lee y Chin-Teng Lin, "*Neural Fuzzy Systems*", Prentice Hall, EE.UU., (1996).

3 C.A. Coello Coello, "Constraint-handling using an evolutionary multiobjective optimization technique", *Civil Engineering and Environmental Systems*, Vol. **17**, pp. 319–346, (2000).

4 A. Florian, "An efficient sampling scheme: updated latin hypecube sampling", *Probabilistic Engineering Mechanics*, Vol. **7**, pp. 123–130, (1992).

5 D.E. Huntington y C.S. Lyrintzis, "Improvements to and limitations of latin hypercube sampling", *Probabilistic Engineering Mechanics*, Vol. **13**, Nº 4, pp. 245–253, (1998).

6 "*Neural Network FAQ*", Newsgroup: comp.ai.neural-nets `ftp://ftp.sas.com/pub/neural/FAQ.html`, W.S. Sarle (Ed.), (1999).

7 D. Frangopol, "Reliability-based structural design", in "*Probabilistic Structural Mechanics Handbook*", Chapman & Hall, C. Sundararajan (Ed.), EE.UU., (1995).

8 A.R. Barron, "Universal approximation bounds for superpositions of a sigmoidal function", *IEEE Trans. Inform. Theory*, Vol. **IT-39**, p. 930, (1993).

9 K.M. Hornick, H. Stinchcombe, H. White y P. Auer, "Degree of approximation results for feedforward networks approximating unknown mapping and their derivatives", *Neural Computation*, Vol. **6**, p. 1262, (1994).

10 D.E. Goldberg, "*Genetic algorithms in search, optimization and machine learning*", Addison-Wesley, New York, (1989).

11 M. Beale y H. Demuth, "*Neural network toolbox for use with MATLAB*", User's Guide, Version 3.0, The MathWorks, Inc., (1997).

12 T. Bäck a y H.P. Schwefel, "Evolution strategies", in "*Genetic algorithms in engineering and computer science*", Wiley, Chichester, U.K., G. Winter, J. Périaux, M. Galá y P. Cuesta (Eds.), (1995).

13 T. Bäck, F. Hoffmeister y H.P. Schwefel "A survey on evolution strategies", *Proceeding of the Fourth International Conference on Genetic Algorithms*, pp. 2–9, R.K. Belew y L.B. Booker (Eds.), Morgan Kauffman Publishers, San Mateo, California, (1991).

14 T.T. Binh y U. Korn, "Scalar optimization with linear and non-linear constraints using evolution strategies", in "*Lecture Notes in Computer Science*", pp. 381–392, B. Reusch (Ed.), Springer-Verlag, Abril, (1997).

15 T. Bäck, "Evolution strategies: an alternative evolutionary algorithm", in "*Artificial Evolution*", J.M. Alliot, E. Lutton, E. Ronald, M. Shoenhauer y pp. 3–20, D. Snyers (Eds.), Springer-Verlag, Berlin, (1996).

16 E. Zitzler, "Evolutionary algorithms for multiobjective optimization: methods and applications", *Tik-Schriftenreihe*, Nº 30, Swiss Federal Institute of Technology Zurich, (1999).

17 J.T. Richardson, M.R. Palmer, G. Liepins y M. Hilliard, "Some guidelines for genetic algorithms with penalty functions", *Proceedings of the Third International Conference on Genetic Algorithms*, pp. 191–197, J.D. Schafer (Ed.), Morgan Kauffman Publishers, George Mason University, (1989).

18 D. Hopkins Loughlin, "*Genetic algorithm-based optimization in the development of tropospheric ozone control strategies*", North Carolina State University, (1998).

APÉNDICE

REDES NEURONALES

Una neurona biológica real está compuesta de una celda y varias dendritas que la unen con otras neuronas. La actividad de cada una abarca las siguientes fases: (a) la recepción de impulsos eléctricos enviados a ella; (b) la transformación de los impulsos recibidos y (c) el envío de esta información transformada a otras neuronas. En la imitación algorítmica de este proceso empleada en el campo de la Inteligencia Artificial, los pasos anteriores se convierten en los siguientes:

a) La información enviada por la neurona l a la neurona m x_{lm} se multiplica por un factor de peso w_{lm} que representa la importancia de aquella en la actividad de ésta. A continuación se suma la información total que llega a la neurona m desde todas las neuronas conectadas con ella

$$y_m = \sum_{l=1}^{L} w_{lm} x_{lm} \tag{1}$$

b) Se transforma el resultado por medio de una función no lineal

$$\tilde{y}_j = f(y_j)$$

Para este propósito la función más usada es, quizás, la llamada *función sigmoide unipolar*, dada por

$$f(x) = \frac{1}{a + \mathrm{e}^{-\gamma x}} \tag{2}$$

c) La información transformada se envía a otras neuronas k, afectando cada recorrido por el factor de peso correspondiente w_{jk}.

De esta manera resulta posible construir un perceptor artificial completo al agrupar las neuronas en estratos, de la manera que muestra la Figura 4, la cual corresponde a un tipo de red de amplia aplicación conocido como *perceptor de tres estratos*. Este dispositivo se utiliza en general para aprender las relaciones de entrada–salida de sistemas complejos de diversas clases. El proceso de entrenamiento es el siguiente: se presenta a la red la información de un conjunto de datos a través de la primera capa en forma normalizada. En este paso, las conexiones w_{ij}, que son el objetivo del entrenamiento, toman usualmente valores aleatorios. Después de sumar la información recibida y aplicar la transformación no lineal en cada neurona de la capa intermedia (u *oculta*) se determinan los valores de salida en la capa del mismo nombre, aplicando las mismas operaciones de suma y transformación. Estos resultados se comparan con los reales, con los cuales hay, necesariamente, una diferencia, la cual se utiliza para actualizar los valores de las conexiones. El proceso se repite para el siguiente conjunto de datos de entrada–salida hasta agotar la base de datos disponible. Lo normal es que al final de este proceso, llamado *época*, el error no se encuentre dentro de lo deseable, por lo cual se deba repetir de nuevo el proceso para múltiples épocas hasta que se alcance dicho objetivo.

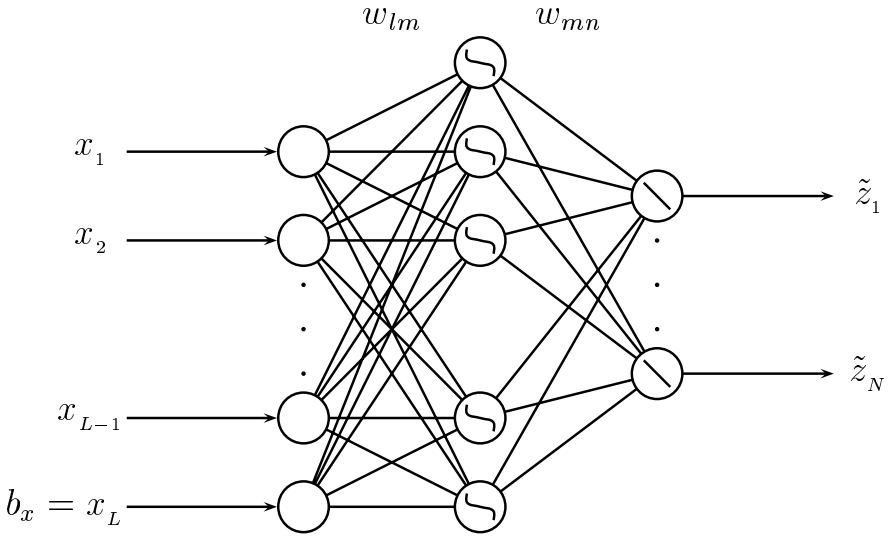

Figura 4. Red neuronal multicapa

Este algoritmo se conoce con el nombre de *propagación hacia atrás*. Los siguientes son los pasos más relevantes del entrenamiento de acuerdo con uno de los métodos más usados, llamado la *regla delta*[3]:

1. Se aplica un vector de datos $[x_1, x_2, \ldots, x_L]$ en las L neuronas del primer estrato. Usualmente se introduce una neurona adicional en esta capa, llamada de sesgo, cuyo valor típico es $x_{L+1} = 1$.

2. Se calcula la información que llega a todas las neuronas del segundo estrato de acuerdo con la ecuación (2). En particular

$$y_m = \sum_{l=1}^{L} w_{lm} x_l + \bar{w}_m \tag{3}$$

donde $\bar{w}_m$ es el peso de la neurona de sesgo.

3. Se aplica la transformación no lineal a la entrada total de cada neurona m del segundo estrato, $m = 1, 2, \ldots M$

$$\tilde{y}_m = f_m(y_m) \tag{4}$$

En esta ecuación el subíndice de la función insinúa la posibilidad de tener diferentes funciones de transformación para cada neurona, lo cual, empero, no es el caso corriente.

4. Se calcula la entrada y la salida de la tercera capa de manera semejante a lo hecho en la segunda

$$z_n = \sum_{m=1}^{M} w_{mn} y_m + \bar{w}_n \tag{5}$$

$$\tilde{z}_n = f_n(z_n) \tag{6}$$

5. El error de estimación que media entre el dato real de cada neurona n, r_n y el estimado por la red es

$$\delta_n = r_n - \tilde{z}_n \tag{7}$$

En la regla delta se busca minimizar el error total cuadrático de la capa de salida, dado por

$$\epsilon = \frac{1}{2} \sum_{n=1}^{N} \delta_n^2 \tag{8}$$

con respecto a los pesos w_{mn}. Esto implica calcular la derivada parcial de la función de error con respecto a w_{mn}, lo que da

$$\frac{\partial \epsilon}{\partial w_{mn}} = -(r_n - \tilde{z}_n) \frac{\partial f_n(z_n)}{\partial z_n} \frac{\partial z_n}{\partial w_{mn}} \tag{9}$$

En el caso de la sigmoide unipolar, la derivada parcial contenida en el lado derecho de la ecuación anterior es

$$\frac{\partial f(x)}{\partial x} = \frac{\mathrm{d} f(x)}{\mathrm{d} x} = \gamma f(x)(1 - f(x)) \tag{10}$$

Esto implica que

$$\frac{\partial f_n(z_n)}{\partial z_n} = \gamma \tilde{z}_n (1 - \tilde{z}_n) \tag{11}$$

$$\frac{\partial z_n}{\partial w_{mn}} = \frac{\partial}{\partial w_{mn}} \sum_{m=1}^{M} w_{mn} y_m + \bar{w}_n = y_m \tag{12}$$

Por tanto, el gradiente de la función de error es

$$\frac{\partial \epsilon}{\partial w_{mn}} = -(r_n - \tilde{z}_n)\gamma \tilde{z}_n (1 - \tilde{z}_n) y_m \tag{13}$$

Como el error mínimo se encuentra en la dirección negativa del gradiente, los pesos de la capa de salida en el paso $[k+1]$ de entrenamiento se calculan añadiendo una porción del gradiente negativo a los pesos calculados en el paso k

$$w_{mn}[k+1] = w_{mn}[k] + \eta(r_n - \tilde{z}_n)\gamma \tilde{z}_n (1 - \tilde{z}_n) y_m \tag{14}$$

En esta ecuación η es el llamado *factor de velocidad de aprendizaje*, cuyo valor se escoge usualmente en el rango $(0, 1)$. Se puede agregar un factor adicional α, llamado de *momentum*, para acelerar el aprendizaje.[4]

6. La actualización de los pesos de la capa oculta se efectúa de manera similar

$$w_{lm}[k+1] = w_{lm}[k] + \eta\gamma \tilde{y}_m (1 - \tilde{y}_m) x_l \sum_{n=1}^{N} (r_n - \tilde{z}_n)\gamma \tilde{z}_n (1 - \tilde{z}_n) w_{mn}[k] \tag{15}$$

7. Se repite el proceso para nuevos valores de entrada–salida (x_l, r_n), $l = 1, 2, \ldots L, n = 1, 2, \ldots, N$ durante varias épocas hasta que el error ϵ esté dentro de lo admisible.

8. Una vez entrenada la red se puede usar para estimar la salida correspondiente a entradas no presentadas previamente por medio de las ecuaciones (3) a (6).

   Existen muchas variantes del algoritmo anterior así como varias propuestas de funciones de activación, las cuales pueden ser ensayadas para mejorar la convergencia.

ESTRATEGIAS EVOLUTIVAS

La implementación de las estrategias evolutivas (EE) se describe ampliamente en la referencia 7. A continuación se presenta un resumen de lo allí expuesto.

En una estrategia evolutiva $(\mu+\lambda)$, μ padres crean λ descendientes por medio de recombinaciones y mutaciones; los mejores μ padres más sus descendientes se seleccionan de manera determinista para volverse padres de la nueva generación. Cada individuo $\vec{a} = (\vec{x}, \vec{\sigma}, \vec{\alpha})$ está compuesto por tres partes: un conjunto de variables objetivo $\vec{x} = (x_1, \ldots, x_n) \in^n$, el cual es un punto en el espacio de búsqueda y es la parte del individuo que se evalúa en la función objetivo, un vector de desviaciones estándar $\vec{\sigma} = (\sigma_1, \ldots, \sigma_{n_\sigma}) \in \mathbb{R}_+^{n_\sigma}$ y un vector de ángulos de inclinación $\vec{\alpha} = (\alpha_1, \ldots, \alpha_{n_\alpha}) \in [-\pi, \pi]^{n_\alpha}$. Los vectores $\vec{\sigma}$ y $\vec{\alpha}$ definen una matriz de covarianza de una distribución normal para cada individuo que se utiliza para definir las mutaciones. Una población de k individuos en la generación t se puede

denotar como $P^{(t)} = \{\vec{a_1}, \vec{a_2}, \ldots, \vec{a_k}\} \in \mathcal{I}^k, k \in \{\mu, \lambda\}$, entendiéndose como un conjunto de elementos de $\mathcal{I}$. La presencia de duplicados en la población es permitida.

La estrategia evolutiva se puede resumir en el siguiente algoritmo:

Algoritmo 1

$t = 0$;
inicialice $P^{(0)} = \{\vec{a}_1, \vec{a}_2, \ldots, \vec{a}_\mu\} \in \mathcal{I}^\mu$;
eval $\{f(x_1), f(x_2), \ldots, f(x_\mu)\}$;
mientras $(t < t_{\max})$
    $\tilde{P} = 0$;
    para $i = 1$ **hasta** λ
        $(\tilde{x}, \tilde{\vec{\sigma}}, \tilde{\vec{\alpha}}) = \mathbf{mut}(\mathbf{rec}(P^{(t)}))$;
        eval $f(\tilde{x})$;
        $\tilde{P} = \tilde{P} \cup (\tilde{x}, \tilde{\vec{\sigma}}, \tilde{\vec{\alpha}})$;
    fin para
    $P^{(t+1)} = \mathbf{seleccione}(\mu, \tilde{P} \cup P^{(t)})$;
    $t = t + 1$;
fin mientras;

En el anterior seudocódigo el operador de recombinación **rec** se define como

$$\mathbf{rec}(P^{(t)}) = \mathbf{co}(\mathbf{se}(P^{(t)})) \tag{16}$$

donde el operador selección **se** escoge aleatoriamente ϱ vectores padres de $P^{(t)}$, $1 \leq \varrho \leq \mu$. Posteriormente sobre estos ϱ vectores padres se aplica el operador combinación **co**. Existen básicamente cuatro tipos de combinación:

- $\omega = 0$: sin recombinación: se utiliza cuando $\mu = 1$ o $\varrho = 1$. En este caso el individuo seleccionado $\vec{k}_i$ se retorna sin modificación

$$\vec{b'} = \vec{k}_i \tag{17}$$

- $\omega = 1$: recombinación global intermedia: aquí el i-ésimo componente de $\vec{x}$ se promedia sobre todos los ϱ padres para obtener el correspondiente componente del vector de la descendencia

$$b'_i = \frac{1}{\varrho} \sum_{k=1}^{\varrho} b_{k,i} \tag{18}$$

- $\omega = 2$: recombinación local intermedia: para cada uno de los individuos de la descendencia, de los ϱ padres se toman dos al azar (k_1 y k_2) para cada componente del vector de los hijos

$$b'_i = u_i b_{k_1,i} + (1 - u_i) b_{k_2,i} \tag{19}$$

  donde $u_i \in [0, 1)$ es un número aleatorio uniformemente distribuído.

- $\omega = 3$: recombinación discreta: aquí, cada componente del vector de la descendencia se copia del correspondiente componente de un vector de padre k_i, escogido aleatoriamente

$$b'_i = b_{k_i,i} \tag{20}$$

Como cada individuo está compuesto por tres parámetros, el operador recombinación **rec** se redefine como

$$\mathbf{rec}(P^{(t)}) = \mathbf{co}_x(\mathbf{se}_x(P^{(t)})) \times \mathbf{co}_\sigma(\mathbf{se}_\sigma(P^{(t)})) \times \mathbf{co}_\alpha(\mathbf{se}_\alpha(P^{(t)})) \tag{21}$$

donde se aplican los siguientes parámetros para las recombinaciones: para x, $\omega_x = 3$ con $\varrho_x = \mu$; para $\vec{\sigma}$, $\omega_\sigma = 2$ con $\varrho_\sigma = \mu$, y para $\vec{\alpha}$, $\omega_\alpha = 0$ con $\varrho_\alpha = 1$.

Cada individuo del algoritmo evolutivo lleva los llamados parámetros de autoadaptación; éstos representan una distribución n-dimensional para mutar al individuo (son $\vec{\sigma}$ y $\vec{\alpha}$ y representan respectivamente las varianzas y las covarianzas de la distribución).

En el ejemplo del artículo se utilizó una matriz de covarianza completa para cada individuo, es decir, $n_\sigma = n$ y $n_\alpha = n(n-1)/2$.

De acuerdo a la estructura generalizada de cada individuo, el operador mutación **mut** se define como

$$\mathbf{mut} = \mathbf{mu}_x(x, \mathbf{mu}_\sigma(\vec{\sigma}), \mathbf{mu}_\alpha(\vec{\alpha}) \tag{22}$$

El operador $\mathbf{mu}_\sigma$ muta el vector $\vec{\sigma}$ recombinado

$$\tilde{\vec{\sigma}} \equiv \mathbf{mu}_\sigma(\vec{\sigma}) = (\sigma_1 \exp(z_1 + z_0), \ldots, \sigma_{n_\sigma} \exp(z_{n_\sigma} + z_0)) \tag{23}$$

donde $z_0 \sim \mathbf{N}(0, \tau_0^2)$, $z_i \sim \mathbf{N}(0, \tau) \forall i \in \{1, \ldots, n_\sigma\}$. Para evitar que las desviaciones estándar se vuelvan prácticamente zero, el valor mínimo de todas las σ_i se debe restringir a ε_σ. Por su parte, el operador $\mathbf{mu}_\alpha$ muta el $\vec{\alpha}$ recombinado

$$\tilde{\vec{\alpha}} \equiv \mathbf{mu}_\alpha(\vec{\alpha}) = (\alpha_1 + z_1, \ldots, \alpha_{n_\alpha} + z_{n_\alpha}) \tag{24}$$

donde $z_i \sim \mathbf{N}(0, \beta^2) \forall i \in \{1, \ldots, n_\alpha\}$. Estos ángulos de rotación se deben mantener en el intervalo $[-\pi, \pi]$, por medio de un mapeo adecuado.

El operador $\mathbf{mu}_x$ muta el $\vec{x}$ recombinado de acuerdo a los $\tilde{\vec{\alpha}}$ y $\tilde{\vec{\sigma}}$ previamente mutados

$$\mathbf{mu}_x(x) = (x_1 + cor_1(\vec{\sigma}, \vec{\alpha}), \ldots, x_n + cor_n(\vec{\sigma}, \vec{\alpha})) \tag{25}$$

donde $\vec{cor} = \mathbf{T}\vec{z}$ es un vector aleatorio, siendo $\vec{z} = (\vec{z}_1, \ldots, \vec{z}_{n_\sigma}), z_i \sim \mathbf{N}(0, \tilde{\sigma}_i^2) \forall i \in \{1, \ldots, n_\alpha\}$ y

$$\mathbf{T} = \prod_{p=1}^{n_\sigma - 1} \prod_{q=p+1}^{n_\sigma} \mathbf{T}_{pq}(\tilde{\alpha}_j) \tag{26}$$

con $j = (2n_\sigma - p)(p+1)/2 - 2n_\sigma + q$. Las matrices de rotación $\mathbf{T}_{pq}(\tilde{\alpha}_j)$ son matrices idénticas excepto que $t_{pp} = t_{qq} = \cos(\alpha_j)$ y $t_{pq} = -t_{qp} = \sin(\alpha_j)$.

En este artículo se ha hecho uso en particular del algoritmo EVOSLINOC (EVOlution Strategy for scalar optimization with LInear and NOnlinear Constraints), descrito ampliamente en la referencia 13. A continuación se presenta un resumen de este método.

En forma general, el problema de optimización a resolver es

$$\min_x f(x) \tag{27}$$

donde $\vec{x} = (x_1, \ldots, x_n) \in \mathcal{F} \subseteq \mathcal{S}$. Se define el *espacio de búsqueda* $\mathcal{S}$ como un rectángulo en el espacio n-dimensional $\mathbb{R}^n$ según

$$x_i^{(\text{inferior})} \leq x_i \leq x_i^{(\text{superior})}, \qquad \forall i = 1, \ldots, n \tag{28}$$

Por otra parte, la *región viable* $\mathcal{F} \subseteq \mathcal{S}$ se define por un conjunto de m condiciones ($m \geq 0$)

$$g_j(x) \leq 0, \qquad \forall j = 1, \ldots, m \tag{29}$$

Para evaluar la aptitud de un individuo se utilizan los siguientes criterios:

- El valor de la función objetivo $f(\vec{x})$, llamada también aptitud $-\mathcal{F}$.
- El grado de violación de las condiciones de las violaciones, conocida como aptitud $-\mathcal{C}$.

La aptitud $-\mathcal{C}$ se calcula como la distancia de Minkowski entre el origen del espacio de condiciones y el punto $(c_1(\vec{x}), c_2(\vec{x}), \ldots, c_m(\vec{x}))$

$$\mathcal{C}(\vec{x}) = \left(\sum_{j=1}^{m} [c_j(\vec{x})]^p \right)^{\frac{1}{p}}; \qquad (p > 0) \tag{30}$$

donde

$$c_j(x) = \max\{g_j(x), 0\}, \qquad \forall j = 1, \ldots, m \tag{31}$$

Usando este concepto, los individuos viables tienen una aptitud $-\mathcal{C} = 0$ y los no viables aptitud $-\mathcal{C} > 0$. De este modo, un individuo $\vec{a}$ se puede representar en la EE EVOSLINOC como $\vec{a} = (x, \vec{\sigma}, \vec{\alpha}, f(x), \mathcal{C}(x))$.

En la población pueden convivir simultáneamente individuos viables y no viables. Para seleccionar cuáles son los mejores individuos en la población, se utilizan los siguientes criterios:

1. Entre dos individuos no viables, el mejor es aquel con una aptitud $-\mathcal{C}$ menor.
2. Entre dos individuos con una aptitud $-\mathcal{C}$ igual, se prefieren aquellos con una aptitud $-\mathcal{F}$ menor.
3. A los individuos no viables con $0 < \mathcal{C} < \mathcal{C}_{extra}$, se les reasigna su aptitud $-\mathcal{C}$ a $\mathcal{C}_{extra}$, de modo tal que estos individuos queden dentro de la misma clase.
4. El nicho para los individuos viables se define mediante la aptitud $-\mathcal{N}_f$

$$\mathcal{N}_f(x) = \frac{f(x) - f(x_{\text{mejorf}})}{\|x - x_{\text{mejorf}}\|^{\beta}} \qquad (x \neq x_{\text{mejorf}}) \tag{32}$$

   donde x_{mejorf} y $f(x_{\text{mejorf}})$ son el vector de variables objetivo y la aptitud $-\mathcal{F}$ del mejor individuo viable, respectivamente. Por otra parte, $\beta \in \mathbb{R} > 0$.

5. Cuando no existe un individuo viable en la población, el nicho para los individuos no viables se define mediante la aptitud $-\mathcal{N}_i$

$$\mathcal{N}_i(x) = \frac{C(x) - C(x_{\text{mejori}})}{\|x - x_{\text{mejori}}\|^{\beta}} \qquad (x \neq x_{\text{mejori}}) \tag{33}$$

donde x_{mejori} y $\mathcal{C}(x_{\text{mejori}})$ son el vector de variables objetivo y la aptitud $-\mathcal{C}$ del mejor individuo no viable, respectivamente.

6. Si existe al menos un individuo viable en la población, el nicho para los individuos no viables se define mediante la aptitud $-\mathcal{N}_{if}$

$$\mathcal{N}_{if}(x) = \frac{C(x)}{\|x - \vec{x}_{\text{mejorf}}\|^{\beta}} \tag{34}$$

Al utilizar el algoritmo $(\mu+\lambda)$-EE, al menos la mitad de los padres deben ser los mejores individuos no viables, elegidos según sus aptitudes $\mathcal{N}_i$ y $\mathcal{N}_{if}$ según corresponda, en tanto que el resto de individuos deben ser viables, y se deben elegir según su aptitud $-\mathcal{N}_f$. Estas aptitudes deben tan pequeñas como sea posible.

Para implementar EVOSLINOC sobre la estrategia evolutiva descrita en la referencia 7, se debe seguir el siguiente algoritmo:

Algoritmo 2

$t = 0$;
inicialice $P^{(0)} = \{\vec{a}_1, \vec{a}_2, \ldots, \vec{a}_\mu\} \in \mathcal{I}^\mu$;
eval $\{f(x_1), f(x_2), \ldots, f(x_\mu)\}$;
eval $\{\mathcal{C}(x_1), \mathcal{C}(x_2), \ldots, \mathcal{C}(x_\mu)\}$;
mientras $(t < t_{\max})$
    $\tilde{P} = P^{(t)}$;
    para $i = 1$ **hasta** λ
        $(\tilde{x}, \tilde{\vec{\sigma}}, \tilde{\vec{\alpha}}) = \mathbf{mut}(\mathbf{rec}(P^{(t)}))$;
        eval $f(\tilde{x})$;
        eval $\mathcal{C}(\tilde{x})$;
        $\tilde{P} = \tilde{P} \cup (\tilde{x}, \tilde{\vec{\sigma}}, \tilde{\vec{\alpha}})$;
    fin para
    eval $\mathcal{N}_i(\tilde{P}), \mathcal{N}_f(\tilde{P}), \mathcal{N}_{if}(\tilde{P})$;
    $P^{(t+1)} = \mathbf{seleccione}_i(k_i, \tilde{P}) \cup \mathbf{seleccione}_f(k_f, \tilde{P}) \qquad k_i + k_f = \mu \qquad k_i \leq \lceil \mu/2 \rceil$;
    $t = t + 1$;
fin mientras;

El operador $\lceil x \rceil$ significa el entero mayor o igual que x.