**Anales del Instituto de Ingenieros de Chile, Agosto 97, pp. 83-92.**

# OPTIMIZACIÓN MEDIANTE ALGORITMOS GENÉTICOS

Pablo Estévez Valencia*


## Resumen

Este artículo introduce en forma tutorial los algoritmos evolutivos de búsqueda denominados algoritmos genéticos. Estos algoritmos se inspiran en la mecánica de la selección natural y la genética para evolucionar una población inicial de puntos sucesivamente hacia mejores regiones del espacio de búsqueda. La evolución de la población se realiza mediante la aplicación de operadores genéticos probabilísticos de selección, recombinación y mutación. Los algoritmos genéticos requieren conocer solamente el valor de la función objetivo en la población de puntos y no sus derivadas. Esto último permite abordar una gran variedad de problemas de optimización no tratables mediante métodos basados en gradientes.

Se describe en detalle un algoritmo genético simple con aplicación a la optimización de parámetros continuos, incluyendo diversos métodos de selección : proporcional, ranking y torneo. La influencia del método de selección en la velocidad de convergencia del algoritmo genético se ilustra mediante un ejemplo de una función continua con múltiples máximos.

Se introducen los algoritmos de nichos paralelos que permiten extender los algoritmos genéticos al caso en que se requiera localizar y mantener múltiples soluciones dentro de una población. Se describen dos algoritmos de nichos paralelos : compartimiento y hacinamiento. Entre estos, el método de hacinamiento presenta el menor costo computacional y además no requiere del ajuste de parámetros adicionales. Se presentan resultados del algoritmo de hacinamiento para dos funciones conteniendo múltiples máximos. En ambos casos el algoritmo encuentra y mantiene todos los óptimos globales.

Finalmente, se considera el problema de optimización con restricciones. Este problema se trata agregando un término de penalización de las restricciones violadas en la función objetivo. Se ilustra un par de ejemplos en los que se utiliza un algoritmo de hacinamiento con penalización. Los resultados indican que los operadores genéticos convencionales de recombinación y mutación pierden efectividad cuando la solución global se encuentra en la frontera del espacio de soluciones factibles. En este caso operadores genéticos alternativos tales como el crossover binomial obtienen mejores resultados.


* Ingeniero Civil Electricista (U. de Chile), Ph.D. en Ingeniería (U. de Tokio).
Académico, Departamento de Ingeniería Eléctrica, Facultad de Ciencias Físicas y Matemáticas, Universidad de Chile.
e-mail : pestevez@tamarugo.cec.uchile.cl

## 1. Introducción

Durante las últimas décadas ha habido un creciente interés en algoritmos basados en el principio de la evolución (supervivencia del más apto). Entre los algoritmos evolutivos más conocidos se incluyen los algoritmos genéticos [3,5,6,10], programación evolucionaria [4], estrategias evolutivas [2,9] y programación genética [7]. El conjunto de estas técnicas se agrupa bajo el nombre de *computación evolucionaria*.

Los algoritmos evolutivos son métodos robustos de búsqueda, que permiten tratar problemas de optimización donde el objetivo es encontrar un conjunto de parámetros que minimizan o maximizan una función de adaptación (fitness). Estos algoritmos operan con una población de individuos $P(t) = \{x_1^t, ..., x_n^t\}$, para la iteración t, donde cada individuo $x_i$ representa un punto de búsqueda en el espacio de las soluciones potenciales a un problema dado. El desempeño de un individuo $x_i$ se evalua según una función de adaptación $f(x_i)$. Esta función permite ordenar del mejor al peor los individuos de la población en un continuo de grados de adaptación.

La población inicial evoluciona sucesivamente hacia mejores regiones del espacio de búsqueda mediante procesos probabilísticos de a) selección de los individuos más adaptados en la población (a mayor grado de adaptación mayor probabilidad de dejar descendencia) y b) modificación por recombinación y/o mutación de los individuos seleccionados.

La estructura del algoritmo evolutivo básico es la siguiente:

```
comenzar
        t =0
        inicializar P(t)
        evaluar P(t)
        mientras (no condición de término) hacer :
                t = t+1
                seleccionar P(t) a partir de P(t-1)
                recombinar y/o mutar P(t)
                evaluar P(t)
        fin
fin
```

Las principales diferencias de los algoritmos evolutivos con los métodos tradicionales de búsqueda (p.ej. métodos basados en gradientes, aleatorios, exhaustivos, etc.) son las siguientes :

- codificación del conjunto de parámetros (en general no se evolucionan los parámetros directamente sino que su codificación).
- búsqueda en paralelo con una población de puntos.
- uso de función de adaptación directamente (no se requiere de derivadas).
- reglas de transición probabilísticas entre una iteración y otra.

Para encontrar los óptimos globales, los algoritmos de optimización hacen uso de dos técnicas : a) explorar áreas desconocidas en el espacio de búsqueda, y b) explotar el conocimiento obtenido de puntos previamente evaluados. Los algoritmos evolutivos combinan ambas técnicas de un modo eficiente.

Este artículo introduce en forma tutorial los algoritmos genéticos y su aplicación a la optimización de parámetros. En particular se considera la optimización numérica de funciones con y sin restricciones. Se presentan resultados de simulaciones de ejemplos simples que ilustran el comportamiento de los algoritmos y operadores genéticos descritos. Finalmente, se describe brevemente una aplicación de algoritmos genéticos para la selección de entradas a clasificadores basados en redes neuronales.

## 2. Algoritmo Genético Simple

El algoritmo evolutivo básico descrito en la introducción da lugar a una gran variedad de modelos, dependiendo de como se especifiquen la forma de representación de los individuos (codificación) y los operadores genéticos de selección, recombinación y mutación. En esta sección se describe el algoritmo genético original propuesto por John Holland [6], junto a algunas de sus variantes. El algoritmo genético de Holland se fundamenta teóricamente en el teorema de schemata, cuya definición y demostración escapa al propósito de este artículo. El lector interesado en este teorema puede consultar el texto de Goldberg [5].

El algoritmo genético simple aplica a problemas de optimización de parámetros continuos de la forma :

$$min \quad f(x_{k1}, x_{k2}, \cdots, x_{kn}), \qquad (1)$$

$$x_{ki} \in [l_i, u_i] \in \Re, \quad l_i < u_i, \quad \forall i = 1, \cdots, n,$$

donde cada componente $x_{ki}$ tiene un dominio definido por una cota inferior $l_i$ y una cota superior $u_i$, y por lo tanto el espacio de búsqueda es un subconjunto de $\Re^n$.

A continuación se especifica un algoritmo genético simple y se explican brevemente sus fundamentos.

2.1 Representación y Función de Adaptación

Un punto de búsqueda $\vec{x}_k = (x_{k1}, x_{k2}, \cdots, x_{kn})$, se representa mediante una tira binaria (binary string). Cada una de las n componentes del vector $\vec{x}_k$ se codifica en binario usando b bits. Luego las representaciones binarias de cada parámetro se concatenan en una sola tira, obteniéndose individuos de largo $\ell = nb$ bits.

El algoritmo genético considera una población inicial de M tiras binarias de largo $\ell$, generadas aleatoriamente. Para evaluar el desempeño de estos individuos se requiere decodificar cada componente representada en binario al entero correspondiente entre 0 y $2^b$-1 y luego reescalarlo en el intervalo real correspondiente al dominio de esa componente según (1). Durante el proceso evolutivo, el algoritmo genético genera una nueva población de tamaño M a partir de la población actual y evalúa el desempeño de los nuevos individuos. Este mecanismo de codificación implica que en el espacio original continuo sólo se realiza una búsqueda con una grilla de puntos. La precisión de esta búsqueda depende del número de bits utilizados para la codificación de los parámetros.

2.2 Métodos de Selección

El mecanismo de selección permite orientar la búsqueda a aquellos puntos más promisorios, i.e. con la mayor adaptación observada hasta el momento. El operador de selección genera a partir de la población actual una población intermedia del mismo tamaño, reproducicndo con un mayor número de copias a los individuos más aptos y eliminando o asignando un menor número de copias a los individuos menos aptos. El operador de selección no produce puntos nuevos en el espacio de búsqueda, sino que determina qué individuos dejarán descendencia y en qué cantidad en la próxima generación.

El algoritmo genético simple utiliza una regla de supervivencia probabilística. En analogía con un problema de la teoría de juegos (multi-armed bandit problem), John Holland postuló que la estrategia óptima de selección consiste en aumentar exponencialmente el número de copias del mejor individuo observado respecto al peor. Este método se conoce como selección proporcional.

2.2.1 Selección Proporcional

La probabilidad de selección $p_{i,t}$ del i-ésimo individuo en la población P(t) depende de la adaptación relativa de éste con respecto a la población :

$$p_{i,t} = \frac{f_i}{\sum_{j=1}^{n} f_j}, \qquad (2)$$

donde $f_j$ es la adaptación del j-ésimo individuo. El número esperado de copias $N_e$ del i-ésimo individuo en la próxima generación es:

$$N_e[i] = Mp_{i,t} = \frac{f_i}{\bar{f}_t}, \qquad (3)$$

donde $\bar{f}_t$ es la adaptación promedio de la población P(t), y M es el tamaño de ésta.

La fase de selección de un algoritmo genético basado en valores esperados se compone de dos partes :

- determinación de los valores esperados $N_e$
- conversión de los valores esperados a números discretos de descendencia (muestreo)

El algoritmo de muestreo debe mantener una población constante y al mismo tiempo proveer de un muestreo exacto, consistente y eficiente. El algoritmo de muestreo original propuesto por Holland se conoce como el método de la ruleta.

Método de la ruleta :

1. Determinar la suma S de las adaptaciones de toda la población.
2. Relacionar uno a uno los individuos con segmentos contiguos de la recta real [0,S), tal que cada segmento individual sea igual en su tamaño a su grado de adaptación.
3. Generar un número aleatorio en [0,S).
4. Seleccionar el individuo cuyo segmento cubre el número aleatorio.
5. Repetir el proceso hasta obtener el número deseado de muestras.

El método de la ruleta sufre de una dispersión ilimitada, es decir la discrepancia entre el número esperado de copias y el número real obtenido por el método de la ruleta puede ser la máxima posible. El algoritmo óptimo Muestreo Estocástico Universal (SUS) corrige esta situación [2].

Muestreo Estocástico Universal :
SUS es análogo a una ruleta con M punteros igualmente espaciados entre sí, de modo que con un solo lanzamiento se obtienen M ganadores. El método SUS no tiene sesgo y su dispersión es la mínima posible. El algoritmo es como sigue :

```
sum = 0
ptr = rand( ) ∈ [0,1]
for i=1 to M
        sum= sum+Ne[i]
        while(sum > ptr) do
                selectind[i]
                ptr = ptr +1
        end
end
```

donde $N_e[i]$ es el número esperado de copias para el individuo i-ésimo según (3). Debe tomarse en cuenta que SUS puede sólo reducir el error de muestreo pero no eliminarlo completamente.

Por otra parte la determinación del valor esperado mediante (3) es muy sensible a la presencia de un individuo super adaptado en la población actual, pudiendo éste llevarse un elevado número de copias generación tras generación y causar una convergencia prematura del algoritmo a un óptimo local, especialmente para poblaciones pequeñas. Una forma de resolver la convergencia prematura por presencia superindividuos es usar selección por ranking.

2.2.2 Selección por Ranking
El algoritmo de selección por ranking es como sigue :

1. Ordenar la población del mejor individuo (x = 1) al peor (x = M).
2. Asignar un número de copias esperadas según

$$\alpha(x) = \eta^+ - (\eta^+ - \eta^-)\frac{(x-1)}{(M-1)}$$

donde

$$_x\ \alpha(x) = M, \quad 1 \le \eta^+ \le 2, \quad \eta^- = 2 - \eta^+.$$

$\eta^+$: máximo valor esperado (1.1-1.2 recomendado)

$\eta^-$: mínimo valor esperado.

3. Usar muestreo estocástico universal (SUS) para llenar la población.

2.2.3 Selección por Torneo
Este método de selección no se basa en valores esperados y no requiere por lo tanto de un algoritmo de muestreo. El algoritmo es como sigue :

1. Escoger tamaño de torneo q (típicamente q=2).
2. Crear una permutación aleatoria de M enteros.
3. Comparar la adaptación de los próximos q-miembros de la población y seleccionar el mejor.
4. Si se acaba la permutación, generar una nueva permutación.
5. Repetir hasta llenar la población.

Selección por torneos de tamaño $q = 2$ es análogo a la selección por ranking con $\eta^+ = 2$, ya que ambos métodos asignan dos copias al mejor, cero copias al peor y una al promedio.

Los distintos métodos de selección pueden ser analizados en términos de su presión selectiva. Ésta se mide como el inverso del tiempo requerido por el mejor individuo para llenar la población con copias de si mismo, cuando no actúa otro operador genético (takeover time).

Los métodos de selección se ordenan en orden creciente de presión selectiva (para valores estándares de sus parámetros) del siguiente modo : selección proporcional, ranking y torneo [2].

2.3 Recombinación
El operador de recombinación (crossover) es el operador de búsqueda más importante en los algoritmos genéticos. Este es un operador sexuado que intercambia el material genético de un par de padres produciendo descendientes que normalmente difieren de sus padres. La idea central es que segmentos distintos de padres diferentes con alta adaptación deberían combinarse en nuevos individuos que tomen ventaja de esta combinación.

El algoritmo genético explota las regiones con mayor adaptación, ya que generaciones sucesivas de selección y recombinación producen un número creciente de puntos en estas regiones.

El operador de recombinación opera con probabilidad $p_c$ (esto permite que en algunos casos no haya recombinación y se mantengan los padres). Dados p y q un par de padres, de largo $\ell$ bits, se escoge aleatoriamente un punto $k \in \{1, \cdots, \ell - 1\}$ y se intercambian los bits a la derecha de esa posición entre ambos individuos, obteniéndose los descendientes s y v, como se indica a continuación :

$$\vec{p} = (p_1, \ldots, p_{k-1}, p_k, \ldots, p_\ell) \qquad \vec{s} = (p_1, \ldots, p_{k-1}, q_k, \ldots, q_\ell)$$
$$\vec{q} = (q_1, \ldots, q_{k-1}, q_k, \ldots, q_\ell) \qquad \vec{v} = (q_1, \ldots, q_{k-1}, p_k, \ldots, p_\ell)$$

El operador crossover de un punto, descrito más arriba, sufre de un sesgo posicional ya que un bit cercano al extremo derecho de la tira tiene una alta probabilidad de intercambio, mientras que un bit en el extremo izquierdo tiene una baja probabilidad de intercambio. El operador crossover binomial corrige este sesgo, intercambiando bits entre padres sobre una base bit a bit, con probabilidad $p \in [0,1]$ aleatoria, distinta para cada posición.

2.4 Mutación

En el algoritmo genético simple, el operador de mutación juega un papel secundario, invirtiendo ocasionalmente un bit. Tasas de mutación pequeñas garantizan que un individuo no difiera mucho de sus padres en el genotipo (tira binaria). La mutación sirve para evitar la pérdida de diversidad producto de bits que han convergido a un cierto valor para toda la población, y que por tanto no pueden ser recuperados por el operador de recombinación. El operador de mutación invierte cada bit de la tira binaria sobre una base bit a bit con probabilidad $p_m$,

$$m'_{\{p_m\}}(q_1, q_2, \cdots, q_\ell) = (q'_1, q'_2, \cdots, q'_\ell), \qquad \forall i \in \{1, \cdots \ell\},$$

donde $q'_i = \begin{array}{ll} q_i & si \quad r > p_m \\ 1 - q_i & si \quad r \le p_m \end{array}$

y $r \in [0,1]$ uniformemente aleatorio, distinto para cada bit $q_i$.

**Ejemplo 1.**

El algoritmo genético simple descrito anteriormente se implementó en lenguaje C. A modo de prueba se procedió a minimizar la función de Bohachevsky,

$$f(x,y) = x^2 + 2y^2 - 0.3\cos(3\pi x) - 0.4\cos(4\pi y) + 0.7,$$

donde $x, y \in [-1,1]$. Esta función tiene un mínimo global en cero y múltiples mínimos locales. En la Fig. 1.1 se ilustra la función de Bohachevsky invertida.

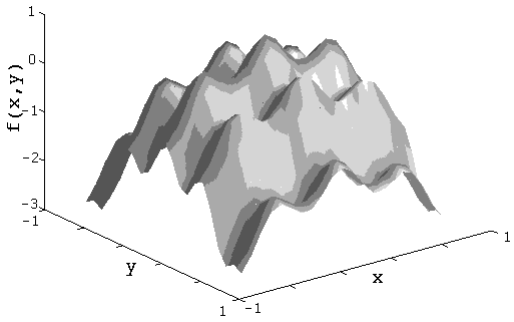


**Fig. 1.1** Función de Bohachevsky invertida.

Los parámetros x e y se codificaron con 16 bits cada uno, resultando en individuos de largo 32. Se consideraron poblaciones de tamaño 100, un número máximo de generaciones de 100 y 30 simulaciones con distintas inicializaciones.

La Fig. 1.2 muestra los resultados promedios del mejor individuo en función del número de generaciones, para los siguientes métodos de selección :

a) Proporcional con ruleta, $p_c$=0.8, $p_m$=0.01.
b) Proporcional con SUS, $p_c$=0.8, $p_m$=0.01.
c) Ranking, $\eta^+ = 1.5$, $p_c$= 0.8, $p_m$= 0.01.
d) Torneo q = 2, $p_c$= 1.0, $p_m$= 0.01.

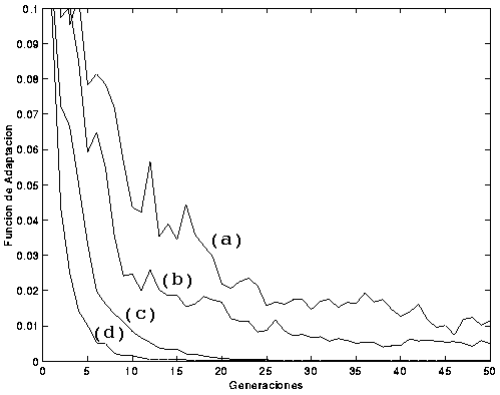


**Fig. 1.2** Comparación de los distintos métodos de selección para la función de Bohachevsky. Las curvas muestran el promedio del mejor individuo en 30 simulaciones con inicializaciones distintas para : a) selección proporcional con ruleta, b) selección proporcional con SUS, c) selección por ranking y d) selección por torneo.

Este ejemplo ilustra que para una mayor presión selectiva se obtiene una mayor velocidad de convergencia. Sin embargo, este resultado no es generalizable al caso de funciones discretas o fractálicas, donde un aumento de la presión selectiva no necesariamente conlleva a mejores resultados [2].

A pesar de la existencia de múltiples mínimos locales en el ejemplo considerado, el algoritmo genético simple encontró el óptimo global, y toda la población convergió a éste.

**3. Algoritmos de Nichos Paralelos**

El algoritmo genético simple converge a una sola solución debido al tamaño finito de la población y a la acumulación de errores estocásticos de muestreo. Los algoritmos de nichos extienden los algoritmos genéticos a dominios que requieren de la localización y mantención de múltiples soluciones, p.ej. la optimización de funciones con múltiples máximos. Los algoritmos de nichos paralelos forman y mantienen subpoblaciones dentro del espacio de una población única, mediante la disminución de la competencia entre puntos distantes en el espacio de búsqueda. Para esto se introduce una métrica d en el espacio genotípico, i.e., $d_{ij} = d(s_i, s_j)$, donde $s_i$ y $s_j$ son las tiras binarias (genotipos) y d es la distancia de Hamming (número de bits diferentes entre tiras). Alternativamente, la distancia se puede medir en el espacio fenotípico, i.e., $d_{ij} = d(x_i, x_j)$, donde $x_i$ y $x_j$ son los vectores de parámetros decodificados y d es la distancia Euclidiana.

Hay dos tipos básicos de algoritmos de nichos : compartimiento (sharing) y hacinamiento (crowding).

Método de Compartimiento

Los algoritmos de compartimiento se basan en la idea de que la adaptación debe ser compartida como un recurso único entre individuos similares en una población. Se define una nueva adaptación compartida $f'$, dividiendo la función de adaptación original por un factor que toma en cuenta la similitud del individuo con el resto de la población :

$$f'(j) = \frac{f(j)}{sh(d_{ij})},$$

donde sh es la función de compartimiento,

$$sh(d) = \begin{cases} 1 - (\frac{d}{\sigma_{share}})^{\alpha}, & d \le \sigma_{share}, \\ 0 & otro \quad caso \end{cases}$$

$\alpha \quad y \quad \sigma_{share}$ son constantes.

La función de compartimiento entrega un 1 cuando dos individuos son idénticos, 0 cuando su grado de disimilitud supera un cierto umbral y un número entre 0 y 1 para grados intermedios de similitud. El umbral de similitud está especificado por la constante $\sigma_{share}$. Si la distancia entre dos individuos es mayor que este umbral entonces ellos no compiten entre sí. La proporción de la población esperada en una clase (p.ej. un máximo local) es función de la adaptación de esa clase dividida por la suma de las adaptaciones de todas las clases. Usualmente se utiliza compartimiento con selección proporcional, pero en principio se podría utilizar cualquier método de selección.

El algoritmo de compartimiento es capaz de mantener múltiples soluciones. Su principal desventaja es que es costoso computacionalmente, ya que se requiere calcular la distancia entre todos los elementos de la población en cada generación. Además requiere del ajuste de los parámetros de la función de compartimiento.

Método de Hacinamiento

Estos algoritmos forman y mantienen nichos mediante el reemplazo de los elementos de la población con individuos similares. En el modelo de hacinamiento determinístico de Mahfoud [8], un individuo compite en un torneo con sus padres. Dado un par de padres y sus descendientes se realizan los torneos que hagan competir a aquellos individuos más cercanos y se selecciona por reemplazo.

Algoritmo de Hacinamiento Determinístico **:**

Repetir n/2 veces

1. Seleccionar 2 padres, $p_1$ y $p_2$, aleatoriamente sin reemplazo.
2. Recombinarlos, dando origen a $c_1$ y $c_2$.
3. Opcionalmente mutar dando origen a $c_1$' y $c_2$'.
4. Dada una medida de distancia d(x,y) aplicar la siguiente regla de reemplazo :

Si $\left[d(p_1,c_1')+d(p_2,c_2')\right] \le \left[d(p_1,c_2')+d(p_2,c_1')\right]$

$\{$Si $f(c_1') < f(p_1)$ reemplazar $p_1$ por $c_1'$

si $f(c_2') < f(p_2)$ reemplazar $p_2$ por $c_2'\}$

De otro modo

$\{$Si $f(c_2') < f(p_1)$ reemplazar $p_1$ por $c_2'$

si $f(c_1') < f(p_2)$ reemplazar $p_2$ por $c_1'\}$

A continuación se ilustran algunos ejemplos de aplicación del método de hacinamiento determinístico. Aunque el método de compartimiento también ha sido implementado no se presentan resultados de este algoritmo, debido a que estos son similares a los de hacinamiento, salvo en lo referente al costo computacional y ajuste de parámetros.

**Ejemplo 2.**
Para probar el algoritmo de hacinamiento determinístico se eligió la función sinusoide atenuada,

$$f_1(X) = e^{-2(\ln 2)\left|\frac{x-0.1}{0.8}\right|} \sin^6(5\Pi x), \qquad 0 \le x \le 1.0 .$$

La variable x se codificó en 30 bits, resultando en tiras de largo 30. Esta función tiene 5 máximos en x=0.1, 0.3, 0.5, 0.7 y 0.9 con alturas 1.0, 0.917, 0.707, 0.459 y 0.250 respectivamente. Para esta función se consideró una población de tamaño 100, probabilidad de crossover 1.0, probabilidad de mutación 0.0 y un número máximo de 500 generaciones.

La fig. 2.1 muestra la distribución inicial de la población de 100 puntos en la función sinusoide atenuada. La fig. 2.2 ilustra la distribución final de la población después de 500 generaciones. El algoritmo de hacinamiento determinístico encuentra y mantiene todos los óptimos de la función sinusoide atenuada.

**Ejemplo 3.**
Como segunda función de prueba se escogió la función en dos dimensiones de Himmelblau :

$$f_2(x,y) = \frac{2186-(x^2+y-11)^2-(x+y^2-7)^2}{2186},$$

$x, y \in [-6,6]$.

Cada parámetro se codificó con 15 bits, resultando en individuos de largo 30. Esta función tiene 4 máximos idénticos en (3,2), (3.584,-1.848), (-3.779,-3.383) y (-2.805, 3.131), de valor 1.0.

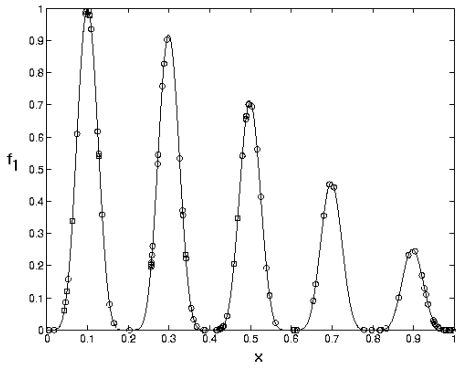


**Fig. 2.1** Distribución inicial de la población de 100 puntos en la función sinusoide atenuada.

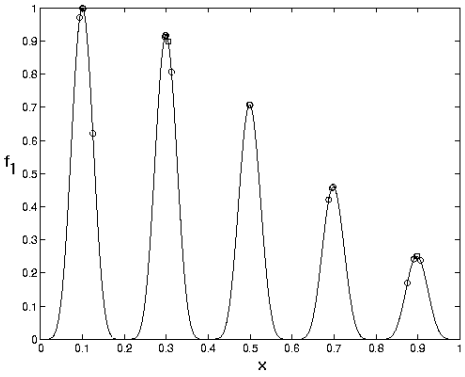


**Fig. 2.2** Distribución final de la población después de 500 generaciones de hacinamiento determinístico. La figura muestra convergencia de la población a todos los máximos de la función sinusoide atenuada.

Para esta función se consideró una población de tamaño 100, probabilidad de crossover 1.0, probabilidad de mutación 0.0 y un número máximo de 100 generaciones.

La fig. 3.1 muestra la distribución inicial de la población de 100 puntos en la función de Himmelblau. La fig. 3.2 ilustra la distribución final de la población después de 100 generaciones. El algoritmo de hacinamiento determinístico encuentra y mantiene los cuatro óptimos de la función.

**4. Optimización con Restricciones**
El problema general de programación no lineal para variables continuas es minimizar o maximizar

$$f(X), \qquad X = (x_1,\ldots,x_n) \in \Re^n ,$$

donde $X \in F \subseteq S$. El conjunto $S \subseteq \Re^n$ define el espacio de búsqueda y el conjunto $F \subseteq S$ define el espacio de soluciones factibles. Hasta el momento se ha considerado $F = S$, es decir optimización sin restricciones.

En esta sección se considera la optimización con restricciones. El espacio de búsqueda S se define como un rectángulo n-dimensional en $\Re^n$ (dominios de variables definidos por cotas superior e inferior) :

$$l(i) \le x_i \le u(i), \qquad 1 \le i \le n,$$

y el conjunto de soluciones factibles F se define por $m \ge 0$ restricciones

$$g_j(X) \le 0, \qquad j = 1, \ldots q,$$
$$h_j(X) = 0, \quad j = q+1, \ldots, m.$$

Hay dos enfoques para manejar individuos no factibles o ilegales : estrategias pro-vida (métodos de reparación y de penalización) y estrategias pro-libre elección (métodos abortivos y anticonceptivos). Mientras que los métodos abortivos eliminan los individuos ilegales que se generen, los métodos anticonceptivos no permiten la generación de individuos no factibles. El método GENOCOP [8] optimiza funciones numéricas con restricciones lineales, utilizando un conjunto de operadores genéticos cerrados que mantienen la factibilidad de las soluciones. Una solución se representa como un vector de números en punto flotante.

Las estrategias pro-vida permiten la presencia de individuos ilegales en una población. Un enfoque es penalizar los individuos no factibles, agregando un término en la función objetivo,

$$eval(X) = \begin{cases} f(X), & si \quad X \in F \\ f(X) + Q(X), & otro \quad caso \end{cases} \qquad (4)$$

donde f es la función objetivo y $Q(X) \ge 0$ es la penalización para el individuo X.
Para construir la función de penalización se utiliza un conjunto de funciones $f_j$, las que miden la violación de la j-ésima restricción,

$$f_j(X) = \begin{cases} max\{0, g_j(X)\}, & si \quad 1 \le j \le q \\ |h_j(X)|, & si \quad q+1 \le j \le m. \end{cases} \qquad (5)$$

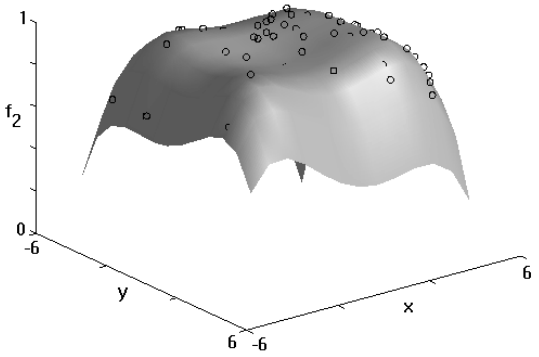


**Fig. 3.1** Distribución inicial de la población de 100 puntos en la función de Himmelblau.

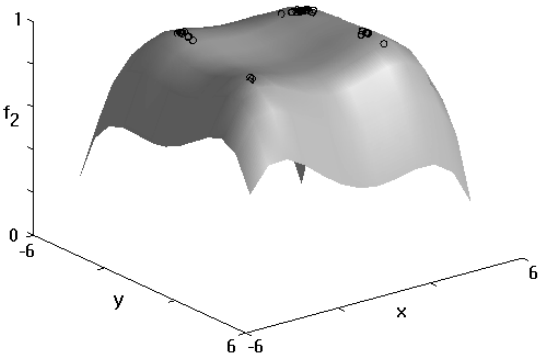


**Fig. 3.2** Distribución final de la población después de 100 generaciones de hacinamiento determinístico. La figura muestra convergencia de la población a todos los máximos de la función de Himmelblau.

En GENOCOP II [8] se considera el problema general de programación no lineal. El método selecciona un punto inicial aleatorio que satisface las restricciones lineales (la población inicial consiste de copias de este único individuo). Los individuos se evaluan según la fórmula

$$eval(X, \tau) = f(X) + \frac{1}{2\tau} \sum_{j=1}^{m} f_j^2(X), \qquad (6)$$

donde $\tau$ es un parámetro de temperatura. Después de algunas generaciones la temperatura disminuye aumentando la presión sobre las soluciones no factibles. La mejor solución encontrada sirve de punto de partida para la próxima generación. El proceso continua hasta que la temperatura alcance el punto de congelación.

**Ejemplo 4.**
Se consideró el mismo algoritmo de hacinamiento determinístico usado en los ejemplos anteriores, pero sujeto a la penalización descrita por ecs. (4-6). Se utilizó el siguiente esquema de enfriamiento de temperaturas $\tau(0) = 1, \quad \tau(t) = 0.5\tau(t-1)$.

Como primera función de prueba se utilizó la función de Michalewicz

$$\begin{aligned} &min \quad f(X) = x_1 x_2^2, \\ &sujeto \quad a \\ &x_1^2 + x_2^2 - 2 \le 0 \end{aligned}$$

Las soluciones globales son : $X_1$*=(-0.832,-1.180), $X_2$*=(-0.832, 1.180) y f(X*) = -1.088. Las variables $x_1$y $x_2$ se codificaron con 30 bits en [-2,2].

La fig. 4.1 ilustra el resultado obtenido después de 100 generaciones de hacinamiento determinístico, utilizando crossover de un punto. El interior del círculo y su frontera corresponden a la región de factibilidad. Se observa que la población converge a la frontera donde se encuentran los óptimos globales, y a un mínimo local en $x_2$=0. Sin embargo, una vez que los puntos están en la frontera el crossover de un punto pierde efectividad. La fig. 4.2 ilustra el mismo caso pero con crossover binomial. Se observa un mejor agrupamiento de la población en torno de los óptimos.

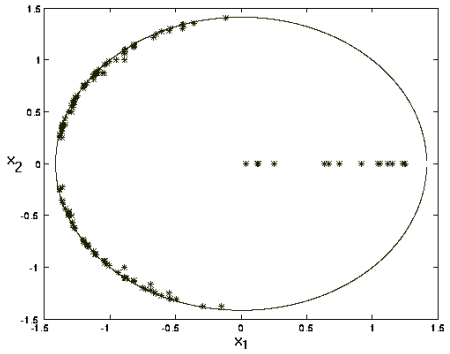


**Fig. 4.1** Distribución de la población en la generación 100 para hacinamiento determinístico con crossover de un punto.

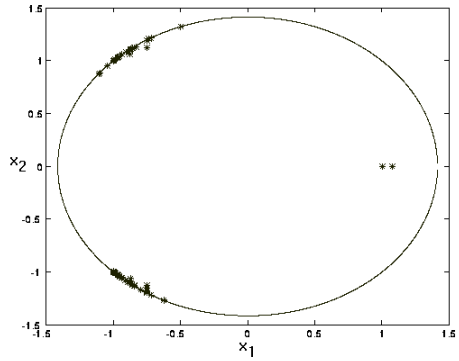


**Fig. 4.2** Distribución de la población en la generación 100 para hacinamiento determinístico con crossover binomial.

**Ejemplo 5**.
Como segunda función de prueba se utilizó la función de Hock

$$\begin{aligned} &\qquad\qquad f_1 = x_2 + 10^{-5}(x_2 - x_1)^2 - 1.0 \quad si \quad 0 \le x_1 \le 2 \\ &min \quad f(X) = \quad f_2 = \tfrac{1}{27\sqrt{3}}((x_1 - 3)^2 - 9)x_2^3 \quad si \quad 2 \le x_1 \le 4 \\ &\qquad\qquad \lfloor \quad f_3 = \tfrac{1}{3}(x_1 - 2)^3 + x_2 - \tfrac{11}{3} \quad si \quad 4 \le x_1 \le 6 \\ &sujeto \quad a \\ &\quad {}^{x1}\!/_{\sqrt{3}} - x_2 \ge 0 \\ &-x_1 - \sqrt{3}x_2 + 6 \ge 0 \ . \\ &0 \le x_1 \le 6, \quad x2 \ge 0 \end{aligned}$$

Esta función tiene tres soluciones globales en $(0,0), \ (3,\sqrt{3}), \ y \ (4,0), \$ en todos los casos $f(X) = -1$. Las variables $x_1$ y $x_2$ se codificaron con 30 bits en el intervalo [-6,6].

Las figs. 5.1 y 5.2 ilustran el resultado obtenido después de 100 generaciones de hacinamiento determinístico, utilizando crossover de un punto y binomial, respectivamente. El interior del triángulo inferior y su frontera corresponden a la región de factibilidad. Nuevamente se observa que una vez que los puntos están en la frontera el crossover de un punto es menos efectivo que el crossover binomial.

**5. Aplicaciones**
Una aplicación en la que se está trabajando actualmente es la clasificación automática de defectos en maderas (nudos, grietas, manchas, etc.) [1]. A partir de las imágenes se realiza una extracción paramétrica de características que sirven de entradas a un clasificador basado en redes neuronales. Es frecuente que el conjunto de características extraídas de la imagen dependa de la experiencia del diseñador y que el número de características no sea óptimo e incluya redundancias. Se están utilizando algoritmos

genéticos para optimizar el conjunto de entradas al clasificador neuronal, lo que es importante para reducir el error de generalización y el tamaño de los modelos. En este caso una tira binaria puede representar las entradas presentes o ausentes, y la evaluación de cada individuo corresponde al desempeño del clasificador neuronal con ese subconjunto de entradas.

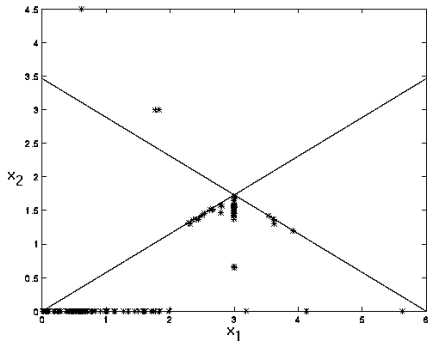


**Fig. 5.1** Distribución de la población en la generación 100 para hacinamiento determinístico con crossover de un punto.

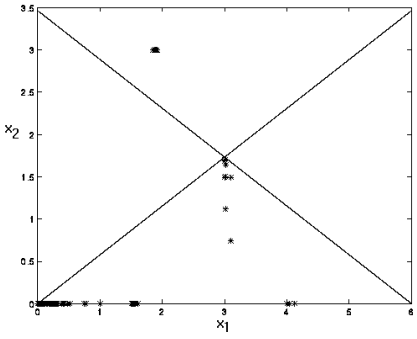


**Fig. 5.2** Distribución de la población en la generación 100 para hacinamiento determinístico con crossover binomial.

## 6. Conclusiones

Se ha realizado una introducción tutorial a los algoritmos genéticos en optimización. Entre los diversos métodos revisados y probados destaca el de hacinamiento determinístico, ya que es barato computacionalmente y además permite localizar y mantener múltiples óptimos. Para problemas con restricciones, sin embargo, se deben considerar operadores genéticos especiales. En casos donde la solución se encuentra en la frontera del espacio de soluciones factibles, la recombinación y la mutación convencional pierden eficacia. El crossover binomial tiene un mejor desempeño en estas circunstancias que el crossover de un punto convencional. Otra alternativa es utilizar operadores de mutación especiales tales como mutación de frontera, donde un punto en la frontera del espacio de soluciones factibles es mutado por otro en esta misma frontera.

**Agradecimientos**

Se agradece especialmente a Rodrigo Caballero Campos por realizar las simulaciones de los ejemplos ilustrados en este artículo. Se agradece también a todos los estudiantes del curso Computación Evolucionaria, dictado por el autor en los años 1996 y 1997, en el Departamento de Ingeniería Eléctrica de la Universidad de Chile.

## Referencias

[1] Aracena, J., Estévez, P., Goles, E., Pérez, C., ¨Reconocimiento de Defectos en Maderas mediante Redes Neuronales¨, XII Congreso Chileno de Ingeniería Eléctrica, Temuco, Nov. 1997 (aceptado).

[2] Back, T., Evolutionary Algorithms in Theory and Practice, Oxford Press, 1996.

[3] Davis, L., Handbook of Genetic Algorithms, Van Nostrand Reinhold, 1991.

[4] Fogel, D., Evolutionary Computation, IEEE Press, 1995.

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

[6] Holland, J.H., Adaptation in Natural and Artificial Systems, MIT Press, Second Edition, 1992.

[7] Koza, J.R., Genetic Programming, MIT Press, 1992.

[8] Mahfoud, S.W., Niching Methods for Genetic Algorithms, Ph.D. Thesis, U. of Illinois at Urbana-Champaign, 1995.

[9] Michalewicz, Z., Genetic Algorithms+Data Structures= Evolution Programs, Springer-Verlag, Second Edition, 1994.

[10] Mitchell, M., An Introduction to Genetic Algorithms, MIT Press, 1996.