Capítulo 11

Computación Evolutiva

11.1 Introducción

La *Computación Evolutiva* (CE) usa modelos computacionales de procesos evolutivos como elementos clave en el diseño e implementación de sistemas que resuelven problemas basados en una computadora. Existe una gran variedad de modelos de CE los cuales han sido propuestos y estudiados en las últimas décadas, y que podemos referencialos comúnmente como *Algoritmos Evolutivos* (AE). Desde el punto de vista de la *Algoritmia*, los AE son algoritmos probabilistas, es decir, toman decisiones aleatorias y se caracterizan fundamentalmente porque el mismo algoritmo puede comportarse de distinta forma aplicado a los mismos datos. Desde el punto de vista de la *Inteligencia Artificial*, los AE son algoritmos heurísticos, es decir, permiten encontrar buenas soluciones en tiempos razonables mediante la evaluación del progreso logrado en la búsqueda de un resultado final. La heurística incorporada en un AE se inspira en la evolución de los seres vivos, retomando los conceptos de selección natural y genética. Así pues, la CE interpreta la naturaleza como una inmensa máquina de resolver problemas y trata de encontrar el origen de dicha potencialidad para utilizarla en nuestros programas.

La CE surge a finales de los años 60 cuando John Holland se planteó la posibilidad de incorporar los mecanismos naturales de selección y supervivencia para resolver una gran cantidad de problemas de Inteligencia Artificial que ya habían sido "resueltos" muy eficientemente por la naturaleza pero que resultaban decepcionantemente inabordables mediante computadoras. El fruto de sus investigaciones se plasmó en el libro *Adaptation in Natural and Artificial Systems* [Holland, 1975], que con el paso del tiempo ha sido considerado como el punto de arranque de la CE. Los intereses de Holland y sus colaboradores fueron en principio académicos, tratando esencialmente de introducir un marco más amplio en el que englobar la Inteligencia Artificial con la intención de resolver ciertos problemas genéricos que constantemente aparecían en dicha disciplina. Sin embargo, a la hora de llevarlas a la práctica, las ideas de Holland resultaban, cuando menos, poco eficientes.

A mediados de los 80, la aparición de computadores de altas prestaciones y bajo costo cambia por completo el panorama. La CE se puede utilizar para resolver con éxito ciertos tipos de problemas de la ingeniería y de las ciencias sociales que anteriormente eran difíciles de tratar. De ésta época son los libros *Genetic Algorithms in Search, Optimization and Machine Learning* [Goldberg, 1989a], y *Handbook of Genetic Algorithms* [Davis, 1991]. En ellos se plantean y resuelven problemas "de la vida real" con los que se empiezan a suscitar intereses económicos.

Así pues, los AE se consolidan con el tiempo como potentes técnicas de búsqueda y optimización, permitiendo asímismo resolver gran cantidad de problemas propios de los procesos de aprendizaje. La clave de su poder reside en que reúnen un conjunto de características que cualquier técnica de búsqueda desearía tener. Los AE son robustos y están concebidos como métodos altamente no lineales de búsqueda global, capaces de resolver problemas de gran dimensionalidad, multimodales, no lineales, en presencia de ruido y dependientes del tiempo, lo que permite tratar problemas muy difíciles si no irresolubles. La búsqueda basada en poblaciones propia de los AE es su característica clave, ya que los sitúa como algoritmos intrínsecamente paralelos, lo cual, junto con los conceptos de selección, herencia y mutación, sienta las bases teorícas del buen funcionamineto de éstos, sintetizadas en el *Teorema del Esquema* [Michalewicz, 1992], el cual básicamente viene a decir que las buenas soluciones exploradas por un AE reciben un incremento exponencial de sus representantes en las subsecuentes generaciones.

Sin embargo, los AE no están exentos de inconvenientes. Fenómenos tales como la *epístasis*, la *decepción*, o el *genetic drift* [Goldberg, 1989a], hacen que la búsqueda caiga en óptimos locales, o impiden una localización diversificada de soluciones en problemas que lo requieren, aspectos éstos que están siendo objeto de intensivos estudios por los investigadores en los últimos años.

Los AE presentan además unas características excelentes de cara a su diseño. Un AE no requiere muchos conocimientos matemáticos del problema para el cual está siendo diseñado. Debido a su naturaleza evolutiva, el AE busca soluciones sin considerar un conocimiento específico del problema, es decir, son métodos *débiles*. No obstante, un AE proporciona una gran flexibilidad para hibridizarse con conocimiento dependiente del problema, lo cual puede mejorar la eficiencia de la técnica en un problema concreto. Un AE puede implementar cualquier clase de función o funciones objetivo y/o restricciones, así como de representar cualquier conjunto de variables de decisión definidas en espacios continuos, discretos, o híbridos.

Un AE típicamente mantiene una *población* de soluciones potenciales (*individuos*) del problema, las cuales evolucionan de acuerdo a reglas de selección y otros operadores, tales como recombinación y mutación. Cada individuo en la población es evaluado, recibiendo una medida de la adecuación (*fitness*) en su entorno. La selección enfoca su atención en los individuos de adecuación alta, explotando así la información de las adecuaciones disponibles. La recombinación y mutación modifican los individuos, proporcionando una heurística general de exploración. Aunque desde un punto de vista biológico estos algoritmos son extremadamente simples, proporcionan sin embargo mecanismos de búsqueda adaptativos muy potentes y robustos.

El Algoritmo 11.1 muestra la estructura de un AE. El algoritmo comienza iniciali-

zando una población, normalmente de forma aleatoria, cuyos individuos son evaluados mediante una función de adecuación que usualmente coincide con la función objetivo del problema de optimización que se quiere resolver. El AE entra en un proceso iterativo que refleja el transcurso de la generaciones. En cada generación se realiza un proceso de selección de los mejores individuos (más adaptados, es decir, con mejor adecuación) los cuales se constituyen como padres para una recombinación y mutación, dando lugar a un nuevo conjunto de individuos $C(t)$ que son asímismo evaluados. Una nueva población es generada mediante un proceso de selección de supervivientes realizado entre la población actual $P(t)$ y los nuevos individuos del conjunto $C(t)$, produciendo una nueva población $P(t+1)$ y una nueva generación. El proceso iterativo termina cuando se cumple una condición de terminación que normalmente se establece como la conclusión de un número prefijado de generaciones.

Algoritmo 11.1 Algoritmo Evolutivo.

```
1: t ← 0; /* generación inicial */
2: inicializar P(t);
3: evaluar P(t);
4: mientras (no se cumpla la condición de terminación) hacer
5:    seleccionar padres de P(t);
6:    recombinar padres y mutar ⇒ C(t);
7:    evaluar C(t);
8:    seleccionar supervivientes de P(t) ∪ C(t) ⇒ P(t+1); /* sustitución generacional */
9:    t ← t+1; /* siguiente generación */
10: fin mientras
```

Generalmente se acepta que un AE debe contener los siguientes cinco componentes:

- Una *representación* apropiada de las soluciones del problema. Este punto es muy importante en el diseño de un AE ya que establece el espacio de búsqueda y de su elección depende la eficiencia del algoritmo. En las siguientes secciones se verán representaciones binarias propias de los Algoritmos Genéticos, así como representaciones *ad-hoc* típicas de determinados problemas de optimización.

- Una forma de crear una *población inicial* de soluciones. Aunque usualmente la población inicial se genera de forma aleatoria dentro del espacio de búsqueda, la incorporación de conocimiento puede ayudar a guiar la búsqueda.

- Una *función de evaluación* capaz de medir la adecuación de cualquier solución, y que hará el papel de "entorno", en el cual las mejores soluciones, esto es, aquellas con mejor adecuación, tengan mayor probabilidad de selección y supervivencia.

- Un conjunto de *operadores evolutivos* que actúan como reglas de transición probabilístas (no deterministas) para guiar la búsqueda, y que combinan entre sí las soluciones existentes con el propósito de obtener otras nuevas.

- El valor de unos *parámetros* de entrada que el AE usa para guiar su evolución (tamaño de la población, número de iteraciones, probabilidades de aplicación de los operadores evolutivos, etc.).

Aunque la CE constituye una gran familia de técnicas entre las que se encuentran las *Estrategias de Evolución* [Rechenberg, 1973], la *Programación Evolutiva* [Fogel y otros, 1966], la *Programación Genética* [Koza, 1972] y los *Algoritmos Genéticos* [Goldberg, 1989a], son estos últimos los que han recibido más atención y han estado sujetos a un mayor estudio tanto teórico como aplicado, por lo que nos dedicaremos a ellos fundamentalmente en este capítulo. La Sección 11.2 describe los elementos fundamentales de un algoritmo genético simple, ilustrándose con un ejemplo de optimización sencillo. La Sección 11.4 muestra algunas técnicas avanzadas usadas en la literatura de cara al diseño eficiente de AE en ambientes de optimización complejos, entre los que se incluyen la presencia de restricciones o la optimización multiobjetivo. Se incluyen también algunas consideraciones generales de cara a la evaluación y validación requeridas en el diseño e implementación de AE. La Sección 11.6 muestra ejemplos de AE para problemas conocidos como el del viajante de comercio y el de selección de variables en Minería de Datos. Finalmente la Sección 11.6 es una propuesta de ejercicio para el diseño, implementación y evaluación de distintos AE para el pasatiempos del Sudoku.

11.2 Un Algoritmo Genético Simple

Los Algoritmos Genéticos (AG) fueron introducidos inicialmente por Holland en 1975 para abstraer y explicar rigurosamente los procesos adaptativos de los sistemas naturales, así como para el diseño de sistemas artificiales de software que retengan los mecanismos importantes de los sistemas naturales. Fué unos años más tarde cuando su alumno, D. Goldberg, implementó el primer AG aplicado en problemas industriales. Estas y otras aplicaciones creadas por estudiantes de Holland convirtieron a los AGs en un campo con base suficiente aceptado para celebrar la primera conferencia en 1985, ICGA´85, la cual se sigue celebrando, actualmente integrada dentro de la Genetic and Evolutionary Computation Conference (GECCO).

Los AG han usado tradicionalmente una representación más independiente del dominio, normalmente, cadenas binarias. Sin embargo, muchas aplicaciones recientes de AG han usado otras representaciones, tales como grafos, expresiones Lisp, listas ordenadas, o vectores de parámetros reales. En esta sección describiremos un AG simple que usa como representación cadenas de bits. El AG que mostramos[1] es el descrito en los trabajos iniciales de Goldberg el cual supuso la lanzadera en el campo de la Computación Evolutiva, tanto a nivel de aplicaciones prácticas, como a nivel teórico, sentando las bases de convergencia al óptimo mediante el Teorema del Esquema.

11.2.1 Representación

En un AG, las soluciones potenciales al problema se representan típicamente mediante cadenas binarias de bits (0's y 1's) de una longitud determinada *long* que vendrá impuesta por el número de variables existentes en la solución y por el número

[1] En [Goldberg, 1989a], Apéndice C, puede encontrarse el código completo en Pascal de un AG simple

de bits necesarios para codificarlas. Otros términos usados a menudo para denominar una solución del problema en un AG son *string* o *estructura*, y, siguiendo el vocabulario de los sistemas biológicos, *cromosoma*. Así, los cromosomas están compuestos por unidades binarias que se denominan *genes*. Al valor de un gen determinado se le denomina *alelo*, y a su posición en el cromosoma *locus*. Al paquete genético total se le denomina *genotipo*, y a la interacción del genotipo con su entorno se le denomina *fenotipo*, que se traduce en la decodificación del cromosoma para la obtención de una solución alternativa (conjunto de parámetros particular, o un punto en el espacio de búsqueda). De esta forma, podemos representar un cromosoma c_i^t en una *generación* (iteración) determinada t como:

$$c_i^t = (b_{i1}^t \dots b_{ilong}^t)$$

con $b_{ij}^t \in \{0,1\}$, $j = 1, \dots, long$. El término *individuo* es frecuentemente utilizado [Goldberg, 1989a] para referirse al conjunto de información *genotipo-fenotipo-adecuación*. Así, podemos representar un individuo X_i^t en una generación t, como la terna:

$$X_i^t = (c_i^t, x_i^t, f_i^t)$$

donde x_i^t es la decodificación (fenotipo) del cromosoma c_i^t, y f_i^t es la adecuación de la solución al entorno o *fitness*.

11.2.2 Obtención de la población inicial

Una población en un AG está formada por un conjunto de m individuos, donde el número m (tamaño de la población) es un parámetro de entrada al AG. De esta forma, en una generación determinada t, podemos representar una población $P(t)$ como:

$$P(t) = \{X_i^t, \dots, X_m^t\}$$

Para obtener una población inicial $P(0) = \{X_i^0, \dots, X_m^0\}$ debemos generar m individuos iniciales. Un procedimiento de inicialización de un individuo X_i^0 consiste simplemente en asignar, para cada gen b_{ij}^0, $j = 1, \dots, long$ de su cromosoma c_i^0, un valor aleatorio 0 ó 1.

Con la decodificación del cromosoma c_i^0 obtendremos su fenotipo x_i^0, y la adecuación de la solución al entorno se obtiene a través de la función de evaluación.

11.2.3 Función de evaluación

Aunque existen otras alternativas, el mecanismo usual de medir la adecuación de una solución consiste simplemente en evaluar su fenotipo a través de la función objetivo f del problema que se esté resolviendo. Así pues, la función de evaluación *eval* se corresponde normalmente con la función objetivo f del problema, y entonces, dado un cromosoma c_i^t y su fenotipo x_i^t, podemos obtener su adecuación f_i^t como:

$$f_i^t = eval(c_i^t) = f(x_i^t).$$

La función objetivo juega un papel fundamental en un AE, puesto que ésta es la información fundamental que se usa del entorno, lo cual hace que estos métodos sean muy generales y robustos.

11.2.4 Selección, muestreo, operadores genéticos y sustitución generacional

Una vez que se han evaluado todas las soluciones de la población en una generación, el proceso evoluciona hacia una nueva generación. La población en la nueva generación t sufrirá una transformación con respecto a la población en la generación anterior $t-1$. Esta transformación se lleva a cabo, en un esquema básico de funcionamiento (Figura 11.1), por medio de la aplicación de las siguientes reglas de transición probabilístas:

1. Selección/muestreo de padres.
2. Cruce.
3. Mutación.
4. Selección de supervivientes/sustitución generacional.

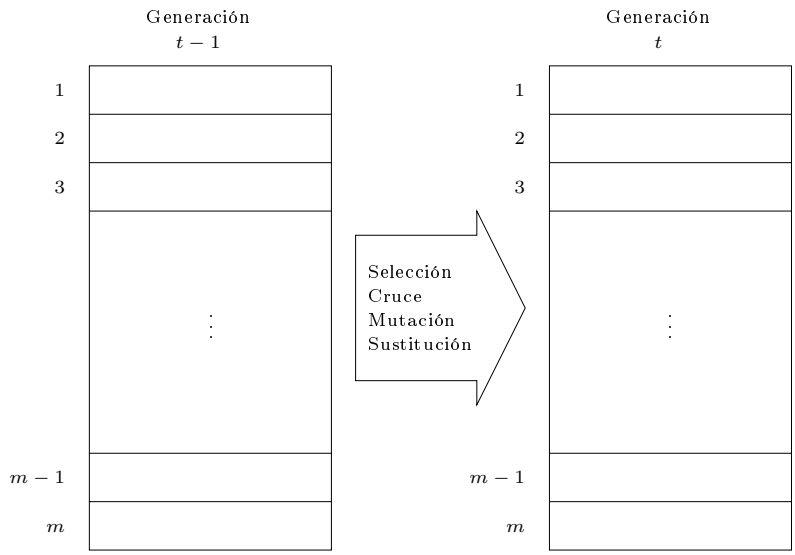

Figura 11.1: Transformación de la población en un AG.

Selección y muestreo El esquema de selección asigna, a cada individuo i de una población en la generación t, el número esperado de descendientes $N_i^t = m \cdot p_i^t$, donde p_i^t es la probabilidad de que el individuo i sea seleccionado. Asociado a un esquema de selección, un algoritmo de muestreo obtiene individuos de la población (números discretos) de acuerdo al número esperado de descendientes de los individuos. En el AG simple que estamos describiendo se usa, como esquema de selección, la *selección proporcional*, y como algoritmo de muestreo, el *muestreo estocástico con reemplazamiento* [DeJong, 1975] o *rueda de ruleta* [Goldberg, 1989a].

Con la selección proporcional se asigna a cada cromosoma de una población en la generación t, la siguiente probabilidad:

$$p_i^t = \frac{f_i^t}{\sum_{i=1}^{m} f_i^t},\ i = 1, \ldots, m.$$

El algoritmo de muestreo estocástico con reemplazamiento realiza los siguientes pasos:

- Calcular la probabilidad acumulada de cada cromosoma:

$$q_i^t = \sum_{j=1}^{i} p_i^t,\ i = 1, \ldots, m;$$

- Repetir m veces los siguientes pasos:
  - Extraer, con reemplazamiento, un número real aleatorio $r \in [0, 1]$;
  - Si $r \leq q_1$ entonces se muestrea el primer cromosoma de la población en la generación t, y en otro caso se muestrea el i-ésimo cromosoma $(2 \leq i \leq m)$ tal que $q_{i-1}^t < r \leq q_i^t$.

Este proceso de muestreo representa una rueda de ruleta con m marcas y separaciones entre ellas fijadas proporcionalmente de acuerdo a la adecuación de los cromosomas.

Operadores genéticos Una vez selccionados un par de padres, el *operador de cruce* se aplica a éstos con probabilidad p_{cruc}, donde p_{cruc} es un parámetro de entrada el cual nos determina el número esperado de cromosomas a los cuales se les aplicará el operador de cruce en cada generación. Con el *cruce simple* [Goldberg, 1989a], se genera un número entero aleatorio $pos \in [1..long-1]$, que indica la posición del punto de cruce. Dados dos cromosomas c_i^t y c_j^t seleccionados como padres en la generación t:

$$c_i^t = (b_{i1}^t \ldots b_{ipos}^t b_{i(pos+1)}^t \ldots b_{ilong}^t)$$

$$c_j^t = (b_{j1}^t \ldots b_{jpos}^t b_{j(pos+1)}^t \ldots b_{jlong}^t)$$

el cruce simple produce los siguientes *hijos*:

$$c'^t_i = (b^t_{i1} \dots b^t_{ipos} b^t_{j(pos+1)} \dots b^t_{jlong})$$

$$c'^t_j = (b^t_{j1} \dots b^t_{jpos} b^t_{i(pos+1)} \dots b^t_{ilong})$$

los cuales contienen información genetica cruzada de ambos padres.

Seguidamente se aplica, a cada hijo, el *operador de mutación*. En el AG que estamos describiendo se usa la *mutación simple*. En cada gen del cromosoma se lleva a cabo un cambio de 0 a 1 o viceversa con probabilidad p_{mut}, parámetro éste de entrada que nos da el número esperado de genes mutados en cada generación.

Sustitución generacional Cuando tenemos un conjunto de descendientes producidos mediante selección-cruce-mutación, el siguiente paso es evaluarlos y decidir qué individuos constituirán la nueva población. Los individuos de la nueva población, se eligirán, en un esquema general como el que muestra el Algoritmo 11.1, de entre la población actual y los descendientes.

En el AG simple que estamos decribiendo se usa el esquema de *sustitución generacional completa*, junto con una estretegia elitista. Con el elitismo, el mejor individuo de la población actual sobrevive en la nueva población. El resto de individuos de la nueva población lo constituyen los descendientes generados via selección-cruce-mutación, hasta formar una nueva población completa de m individuos. Obviamente, los individuos con mayor probabilidad de selección (mayor adecuación) se seleccionarán un número mayor de veces, y tendrán mayor contribución en las subsiguientes generaciones.

El resto de la evolución, como muestra el Algoritmo 11.1, es simplemente un proceso iterativo de selección-cruce-mutación-evaluación-sustitución en el transcurso de las generaciones hasta que se cumple la condición de parada, que suele ser alcanzar un número determinado de generaciones o el cumplimiento de algún predicado de éxito si es posible reconocer soluciones optimales o, en su defecto, satisfactorias para un decisor.

11.2.5 Parámetros de entrada

En cuanto a los parámetros que el AG utiliza, los más significativos son el *tamaño de la población*, el *número de generaciones* , la *probabilidad de cruce* y la *probabilidad de mutación*. De Jong [DeJong, 1975], completando los primeros estudios de Holland, realiza un análisis para la correcta fijación de parámetros haciendo distinción entre situaciones *off-line* y situaciones *on-line*. Para el tamaño de la población, tales estudios apuntan a un buen rendimiento off-line para poblaciones grandes, ya que convergen finalmente mejor debido a una mayor diversidad, y a un buen rendimiento on-line para poblaciones pequeñas, ya que éstas tienen la habilidad de cambiar más rápidamente y ofrecer una convergencia inicial mejor. En [Goldberg, 1989b] se establece una guía para la elección de posibles tamaños de población para representaciones binarias de longitud fija en función de la longitud de los cromosomas. Asímismo, en [Goldberg, 1989b] se estima que el número esperado de generaciones hasta la convergencia es una

función logarítmica del tamaño de la población. Para las probabilidades de cruce y mutación, los estudios realizados en el tema apuntan a probabilidades de cruce altas y probabilidades de mutación bajas. Los siguientes valores han sido considerados como aceptables para un buen rendimiento de los AG (representación binaria) para funciones de optimización:

- Tamaño de la población: 50–100.
- Probabilidad de cruce: 0.60.
- Probabilidad de mutación 0.001.

11.2.6 Ejemplo: Optimización de una función simple

En esta sección consideramos la optimización de una función simple de una variable para ilustrar el funcionamiento del AG simple. Dada la siguiente función [Michalewicz, 1992]:

$$f(x) = x \cdot sin(10\pi \cdot x) + 1{,}0$$

el problema de optimización consiste en encontrar $x \in [-1, 2]$ que maximize la función f, es decir, encontrar x^* tal que:

$$f(x^*) \geq f(x), \ \forall x \in [-1, 2].$$

Representación Asumimos que se requiere una precisión de 6 dígitos después del punto decimal. Puesto que el dominio de la variable x tiene longitud 3, la precisión requerida implica que el dominio $[-1, 2]$ debe ser dividido en $3{\cdot}10^6 = 3000000$ partes de igual tamaño, lo que significa que se requieren 22 bits para representar un cromosoma ($long = 22$), ya que:

$$2097152 = 2^{21} < 3000000 \leq 2^{22} = 4194304.$$

Para decodificar un cromosoma $c_i^t = (b_{i1}^t \ldots b_{i22}^t)$ realizamos los dos siguientes pasos:

1. Calcular la representación en base 10 del cromosoma:

$$x'^t_i = \sum_{j=1}^{22} b_{ij}^t \cdot 10^{22-j};$$

2. Realizar un cambio de escala de acuerdo a los valores límite del dominio de la variable. El fenotipo x_i^t resultante es:

$$x_i^t = -1{,}0 + x'^t_i \cdot \frac{3}{2^{22} - 1}$$

donde -1 es el límite inferior del dominio, y 3 la longitud del dominio.

Los cromosomas 00000000000000000000 y 11111111111111111111 representan los valores $-1{,}0$ y $2{,}0$ respectivamente.

Población inicial y evaluación La población inicial se puede obtener fácilmente generando aleatoriamente m cromosomas de 22 bits, para seguidamente proceder a sus decodificaciones y evaluaciones. La evaluación de un cromosoma c_i^t se obtiene para este ejemplo de la siguiente forma:

$$eval(c_i^t) = f(x_i^t) = x_i^t \cdot sin(10\pi \cdot x_i^t) + 1{,}0 = f_i^t$$

La Tabla 11.1 muestra un ejemplo de 3 individuos (cromosoma, fenotipo, adecuación).

Cromosoma c_i^t	Fenotipo x_i^t	Adecuación f_i^t
1000101110110101000111	0,637197	1,586345
0000001110000000010000	$-0{,}958973$	0,078878
1110000000111111000101	1,627888	2,250650

Tabla 11.1: Tres individios de longitud 22 para el problema de optimización del ejemplo.

Cruce Supongamos que los cromosomas 0000001110000000010000 y 1110000000111 111000101 se han seleccionado como padres. El operador de cruce se aplica establecido un punto de cruce en la sexta posición. La Figura 11.2 muestra los hijos resultantes, con las siguientes evaluaciones:

$$eval(0000000000111111000101) = f(-0{,}998113) = 0{,}940865$$
$$eval(1110001110000000010000) = f(1{,}666028) = 2{,}459245$$

Nótese que el segundo hijo tiene mejor adecuación que sus dos padres.

00000	|	01110000000010000	$\Rightarrow$	00000	|	00000111111000101
		$\updownarrow$				
11100	|	00000111111000101	$\Rightarrow$	11100	|	01110000000010000

Figura 11.2: Cruce simple en la posición 6.

Mutación: Supongamos que operador de mutación se aplica al cromosoma 1110000 000111111000101 Dada la baja probabilidad de mutación, el operador aplica el cambio únicamente en el gen 10 del cromosoma. La Figura 11.3 muestra el descendiente. La evaluación del descendiente es:

$$eval(11100000011111111000101) = f(1{,}630818) = 2{,}343555$$

lo que significa que el individuo ha mejorado tras la mutación.

111000000**0**111111000101 $\Rightarrow$ 111000000**1**111111000101

Figura 11.3: Mutación en el gen 10.

Resultados experimentales El mejor cromosoma obtenido después de 150 generaciones, con $m = 50$, $p_{cruc} = 0{,}25$ y $p_{mut} = 0{,}01$ ha sido:

$$eval(11110011010000100000101) = f(1{,}850773) = 2{,}850227.$$

11.3 Fundamentos de los Algoritmos Genéticos

¿Porqué trabajan bien los AG? Para responder esta pregunta tendríamos que recurrir a todo el formalismo teórico que sustenta a este tipo de algoritmos [DeJong, 1975; Goldberg, 1989a; Holland, 1975; Michalewicz, 1992]. Sin intención de utilizar de forma exhaustiva y rigurosa tal formalismo en un capítulo introductorio como este, podemos, no obstante, realizar algunos comentarios de carácter general sobre los fundamentos de los AG.

En un AG, la búsqueda de soluciones idóneas de un problema consiste en la búsqueda de determinadas cadenas binarias, de forma que el universo de todas las posibles cadenas puede ser concebido como un paisaje imaginario con cimas y valles que albergan a buenas y malas soluciones respectivamente. Tambien podemos definir *regiones* en el espacio de soluciones fijándonos en las cadenas que posean unos o ceros en lugares determinados, y que usualmente son denominadas *esquemas* o *hiperplanos*.

Un esquema es por tanto un patrón de similitud que describe un subconjunto de cadenas con similitudes en ciertas posiciones de éstas, y se define sobre el alfabeto ternario $\{0, 1, *\}$, donde el símbolo $*$ en una posición del esquema indica que el alelo en dicha posición no está determinado. Así por ejemplo, para cadenas de longitud 5, el esquema $(1****)$ representa todas las cadenas que comiencen por 1 en el espacio de posibilidades, y el esquema $(1*0*1)$ describe un subconjunto de cuatro miembros $\{(10001), (10011), (11001), (11011)\}$.

De esta forma, el conjunto de cadenas que forma una población en el AG sondea el espacio de soluciones en muchos esquemas a la vez. Por tanto, una de las claves de la buena conducta de los AG reside en que cada cadena individual pertenece a todos los esquemas en los cuales aparece uno cualquiera de sus bits. Un AG que manipule una población de un número determinado de cadenas está tomando muestras de un número de esquemas enormemente mayor.

J.H. Holland estimó que el número de esquemas procesados útilmente en un AG que manipule un población de m individuos es del orden de m^3, resultado éste importante al que se dio el nombre de *paralelismo implícito*.

El AG explota los esquemas de más alto rendimiento del espacio de soluciones, ya que las sucesivas generaciones de reproducción y cruce generan un número creciente de cadenas pertenecientes a ellas. La mutación, que por sí sola no puede generar progresos en la búsqueda de una determinada solución, proporciona un mecanismo de exploración (nuevos esquemas) que impide el desarrollo de una población uniforme e incapaz de ulterior evolución. Ahora bien, el efecto de los operadores de cruce y mutación puede provocar también que determinadas cadenas abandonen el esquema de sus progenitores en el transcurso de las generaciones.

La probabilidad de que una cadena particular abandone el esquema de sus progenitores depende del *orden* y de la *longitud* de dicho esquema. El orden $o(H)$ se define como el número de posiciones fijas del esquema H, y será útil para calcular la probabilidad de supervivencia del esquema H bajo el efecto de la mutación. La longitud $\delta(H)$ se define como la distancia entre la primera posición fija y la última posición fija en el esquema H, y será útil para calcular la probabilidad de supervivencia del esquema H bajo el efecto del cruce.

Así pues, los esquemas constituidos con pocas y contiguas posiciones fijas que además tengan una adecuación por encima de la media, y que llamamos *bloques constructivos*, tienen mayor probabilidad de salir intactos de cruces y mutaciones y propagarse a las generaciones futuras.

Aplicando únicamente el operador de reproducción, el número esperado de representantes del esquema H en la población en el instante $t+1$ viene dado por la siguiente expresión:

$$N(H,t+1) = N(H,t) \cdot \frac{f(H,t)}{\overline{f^t}}$$

donde $N(H,t)$ es el número de representantes del esquema H en la población en el instante t, $f(H,t)$ es la adecuación media de los representantes del esquema H en la población en el instante t, y $\overline{f^t} = \sum_{i=1}^{m} f_i^t/m$ es la adecuación media de la población en el instante t. Si asumimos que un esquema H permanece por encima de la media en un $\epsilon\,\%$, es decir, $f(H,t) = \overline{f^t} + \epsilon \cdot \overline{f^t}$, entonces:

$$N(H,t+1) = N(H,0)(1+\epsilon)^t$$

lo que significa que los esquemas por encima de la media reciben un incremento exponencial de sus representantes en las siguientes generaciones.

Si consideramos también el efecto de los operadores de cruce y mutación, tenemos que:

$$N(H,t+1) \geq N(H,t) \cdot \frac{f(H,t)}{\overline{f^t}} \left[1 - p_{cruc} \cdot \frac{\delta(H)}{long-1} - o(H)p_{mut}\right]$$

Tal efecto no es significante si el esquema es de longitud corta y de orden bajo, por lo que en tales condiciones se sigue produciendo un incremento exponencial de representantes del esquema en las siguientes generaciones.

Para concluir, podemos resumir lo comentado anteriormente a través de la *Hipótesis del Bloque Constructivo* y del *Teorema Fundamental de los AG* o *Teorema del Esquema*:

Hipótesis del Bloque Constructivo:

Un AG busca soluciones cerca del óptimo a través de la yuxtaposición de esquemas de longitud corta, orden bajo y adecuación alta, llamados bloques constructivos.

Teorema del Esquema:

Los esquemas de longitud corta, orden bajo y adecuación por encima de la media reciben un incremento exponencial de sus representantes en subsecuentes generaciones de un AG.

11.4 Diseño de Algoritmos Evolutivos

Aunque la teoría que sustenta a los AE proporciona una explicación de porqué estos algoritmos convergen a soluciones optimales, en aplicaciones prácticas surgen inconvenientes que hacen que no siempre se cumplan los resultados teóricos. Una de las causas comunes que hacen que los AE se vean incapacitados para encontrar soluciones óptimas en determinadas circunstancias es el fenómeno de la *convergencia prematura*. Tal problema, que aparece también en otras técnicas de optimización, consiste como su nombre indica en una convergencia demasiado rápida, posiblemente a un óptimo local. En CE, esto es debido a la presencia de *superindividuos* en la población, los cuales son mucho mejores que la adecuación media de la población, por lo que acarrean un gran número de descendientes e impiden que otros individuos contribuyan con su descendencia en la siguiente generación, con la consecuente pérdida de información, lo que lleva consigo la formación de poblaciones altamente uniformes e incapaces de evolucionar. No obstante, ciertos diseños de AE son más propensos a la convergencia prematura que otros, y los investigadores más conocidos en el tema aluden a los mecanismos de selección y muestreo, a la ruptura de esquemas debido al cruce, a la fijación de parámetros y a las características propias de la función. Desafortunadamente, la convergencia prematura no es el único problema con el que se encuentran los AE. Aunque los AE están clasificados como algoritmos *débiles*, existe una gran variedad de problemas los cuales resultan difíciles o imposibles de resolver con el esquema simple de AG visto anteriormente, como por ejemplo, problemas que contienen restricciones no triviales, problemas con función multimodal, problemas con múltiples objetivos, etc. Todo esto ha llevado en los últimos años al estudio de mejoras y extensiones sobre los AE que proporcionan un mejor rendimiento de éstos así como un mayor ámbito de aplicación, tales como representaciones alternativas, mecanismos de obtención de poblaciones iniciales, funciones de evaluación, esquemas de selección, muestreo y sustitución generacional, nuevos operadores de cruce y mutación, técnicas basadas en el conocimiento específico del problema, técnicas para el manejo de restricciones, técnicas de diversidad, optimización multiobjetivo, etc. [Goldberg, 1989a][Michalewicz, 1992].

En esta sección describimos algunas de estas mejoras y extensiones, destacables por haber proporcionado buenos rendimientos y por su aplicabilidad en un gran número de entornos de actual relevancia. Destacamos la representación de soluciones, los esquemas de selección, muestreo y sutitución generacional, los operadores de variación, el manejo de restricciones, y la optimización multi-objetivo, como aspectos más siginificativos.

11.4.1 Representación

La elección de la representación de las soluciones del problema adquiere una gran importancia en CE. La representación elegida puede limitar directamente la forma en la que el sistema observa su ambiente, y repercute en gran medida en el diseño de otros componentes del sistema. Existe una gran variedad de alternativas de representación en CE. Tal variedad aparece incluso considerando únicamente alfabetos binarios, encontrándonos con representaciones de longitud fija y de longtud variable, representaciones con codificación binaria y con codificación Gray, representaciones haploides y diploides, etc. [Goldberg, 1989a]. El alfabeto binario facilita los análisis teóricos y permite el diseño de operadores genéticos elegantes. Sin embargo, la representación binaria puede presentar inconvenientes cuando se aplica a problemas numéricos multidimensionales que requieren una alta precisión, o en presencia de restricciones no triviales [Michalewicz, 1992], y puede resultar difícil de aplicar y no natural en muchos problemas prácticos. Por otro lado, el paralelismo implícito no depende del uso de representaciones binarias, por lo que se abre un camino hacia el uso de representaciones distintas. Estas representaciones pueden diferir de la representación con cadenas binarias tanto en la cardinalidad del alfabeto como en la estructura propiamente dicha. Así, se han considerado alfabetos de mayor cardinalidad, como pueden ser los enteros y los reales, que resultan especialmente útiles en problemas de optimización numérica, y una gran variadad de estructuras de datos diferentes, como por ejemplo matrices, permutaciones, listas, árboles, grafos, etc. Aunque la implicación de la representación en el rendimiento de los AG es un tema que sigue aún bastante inexplorado, muchos investigadores [Michalewicz, 1992] mantienen que una representación natural de las soluciones del problema junto con una familia de operadores genéticos aplicables, puede ser bastante útil en la aproximación de soluciones para muchos problemas.

11.4.2 Esquemas de selección, muestreo y sustitución generacional

En el AG simple descrito en la sección anterior se han utilizado la selección proporcional, el muestreo estocástico con reemplazamiento, y la sustitución generacional completa. Con la selección proporcional se tiene el inconveniente de que la presencia de individuos por encima de la adecuación media puede dar lugar a la convergencia prematura. Con el muestreo estocástico con reeplazamiento puede ocurrir que un mismo individuo sea seleccionado un número excesivo de veces (incluso podría completar la nueva población), lo que daría lugar a poblaciones muy uniformes. La sustitución generacional completa puede producir un efecto negativo en la componente de ex-

plotación del AG. Todos estos inconveniente motivaron el estudio de mejoras en el diseño de AG cuyos primeros y más reconocidos trabajos fueron realizados por DeJong [DeJong, 1975]. Aqui describiremos algunas de las técnicas que han tenido un mayor impacto: selección por ranking, selección por torneo, muestreo universal, sustitución steady-state, y el modelo del factor de crowding.

Selección por ranking Con estos métodos, los individuos son ordenados en una lista de acuerdo a sus adecuaciones, y la probabilidad de selección de un individuo se obtiene en base a su posición en la lista ordenada. Así, con el *ranking lineal* la probabilidad de selección de un cromosoma c_i^t se obtiene como:

$$p_i^t = (a_{max} - (a_{max} - a_{min}) \cdot \frac{rank(c_i^t) - 1}{m - 1}) \cdot \frac{1}{m}$$

donde $rank(c_i^t)$ es la posición en la lista ordenada (del mejor al peor individuo) del cromosoma c_i^t, y los coeficientes $1 \leq a_{max} \leq 2$ y $a_{min} = a_{max} - 1$ representan el valor esperado de descendientes del peor y mejor cromosoma de la población respectivamente ($a_{max} = 1{,}2$ recomendado).

Otra posibilidad [Michalewicz, 1992] es utilizar un parámetro q introducido por el usuario y definir la probabilidad de selección mediante la siguiente función no lineal:

$$p_i^t = q \cdot (1 - q)^{rank(c_i^t) - 1}$$

donde $0 < q < 1$ es un parámetro definido por el usuario.

Selección por torneo Un grupo de individuos (2 en el torneo binario) se muestrean uniformemente de la población y el individuo con mejor adecuación del grupo es seleccionado.

Los distintos métodos de selección pueden ser analizados en términos de su *presión selectiva*. Una medida de ésta es el inverso del tiempo requerido por el mejor individuo para llenar la población con copias de sí mismo, cuando no actúa otro operador genético. Los métodos de selección se ordenan en orden creciente de presión selectiva (para valores estándares de sus parámetros), del siguiente modo: selección proporcional, selección por ranking, selección por torneo.

Muestreo estocástico universal La idea básica de este método (Algoritmo 11.2) es construir una rueda de ruleta con m punteros distribuidos equidistantemente, en lugar de un sólo puntero como ocurre con la rueda de ruleta convencional. De esta forma, con un simple "giro" de la ruleta se seleccionan m individuos a la vez.

Sustitución steady-state Como mecanismo de sustitución generacional alternativo a la sustitución generacional completa, podemos destacar la *sustitución de estado estable (steady-state)*. Con esta técnica, se selecciona un número $n \in [1..m]$ de individuos los cuales serán sustituidos por n nuevos. Para la elección de los individuos que mueren suele utilizarse un ranking inverso, es decir, ordenando la lista de cromosmas

Algoritmo 11.2 Muestreo estocástico universal

```
1: sum ← 0;
2: ptr ← rand() ∈ [0,1];
3: para i = 1 hasta m hacer
4:    sum ← sum + N_i^t; /* N_i^t = m · p_i^t */
5:    mientras sum ≥ ptr hacer
6:       seleccionar individuo i;
7:       ptr ← ptr + 1;
8:    fin mientras
9: fin para
```

de menor a mayor adecuación. Este técnica supone una familia de estrategias de sustitución generacional. La sustitución generacional completa pertenece a esta familia con $n = m$. El modelo del factor de crowding es también un caso particular con la particularidad que comentamos a continuación.

Modelo del factor del crowding Un individuo nuevo sustituye a uno viejo el cual es seleccionado de entre un subconjunto de CF miembros elegidos aleatoriamente de la población completa. En este subconjunto se selecciona para morir el individuo más parecido al nuevo, usando como medida un contador de semejanza bit-a-bit.

11.4.3 Operadores de variación

Se han descrito multitud de variantes para los operadores de variación básicos en CE. Aunque muchas variantes se han llevado a cabo en representaciones binarias, es en las representaciones de alta cardinalidad en donde se observa una mayor diversidad de técnicas. Podemos destacar los siguientes operadores:

Operadores de cruce:

- *Cruce uniforme* (rep. binarias y reales). Cada gen en los hijos se crea copiando el correspondiente gen de un padre u otro, utilizando para ello una máscara de cruce generada aleatoriamente. Donde hay un 1 en la máscara, los genes en el primer hijo se toman del primer padre, y donde hay un 0 los genes se toman del segundo padre. Los genes del segundo hijo se establecen con las decisiones inversas.

- *Cruce aritmético* (rep. reales). Dados dos cromosomas $c_i^t = (b_{i1}^t \dots b_{ilong}^t)$ y $c_j^t = (b_{j1}^t \dots b_{jlong}^t)$ seleccionados como padres para la operación de cruce, se generan dos hijos ${c'}_i^t = ({b'}_{i1}^t \dots {b'}_{ilong}^t)$ y ${c'}_j^t = ({b'}_{j1}^t \dots {b'}_{jlong}^t)$, con ${b'}_{il}^t = a \cdot b_{il}^t + (1-a) \cdot b_{jl}^t$, ${b'}_{jl}^t = a \cdot b_{jl}^t + (1-a) \cdot b_{il}^t$, para $l = 1, \dots, long$, donde $a \in [0,1]$ es un número aleatorio.

- *Cruce plano* (rep. reales). Dados dos cromosomas $c_i^t = (b_{i1}^t \dots b_{ilong}^t)$ y $c_j^t = (b_{j1}^t \dots b_{jlong}^t)$ seleccionados como padres para la operación de cruce, se genera

el cromosoma $c_k^t = (b_{k1}^t \dots b_{klong}^t)$, con b_{kl}^t generado aleatoria y uniformemente en el intervalo $[b_{il}^t, b_{jl}^t]$, para $l = 1, \dots, long$.

- *Cruce BLX-α* (rep. reales). Dados dos cromosomas $c_i^t = (b_{i1}^t \dots b_{ilong}^t)$ y $c_j^t = (b_{j1}^t \dots b_{jlong}^t)$ seleccionados como padres para la operación de cruce, se genera el cromosoma $c_k^t = (b_{k1}^t \dots b_{klong}^t)$, con b_{kl}^t generado aleatoria y uniformemente en el intervalo $[min_{kl}^t - I\alpha, max_{kl}^t + I\alpha]$, con $min_{kl}^t = min(b_{il}^t, b_{jl}^t)$, $max_{kl}^t = max(b_{il}^t, b_{jl}^t)$, $I = max_{kl}^t - min_{kl}^t$, para $l = 1, \dots, long$. El cruce plano es un caso particular del cruce BLX-α para $\alpha = 0{,}0$.

- *Cruce PMX, OX y CX.* Usados cuando se utilizan permutaciones como representación (véase Sección 11.6).

Operadores de mutación:

- *Mutación uniforme* (rep. reales). El operador de mutación uniforme mostrado anteriormente para representaciones binarias puede ser definido también para representaciones reales cambiando el gen a mutar por un valor generado aleatoria y uniformemente en el dominio de la variable.

- *Mutación no uniforme* (rep. reales). Dado el cromosoma $c_i^t = (b_{i1}^t \dots b_{ilong}^t)$, y supuesto que el gen b_{il}^t ha sido seleccionado para la mutación, el resultado es $c'^t_i = (b_{i1}^t \dots b'^t_{il} \dots b_{ilong}^t)$, donde b'^t_{il} es, para representaciones reales:

$$b'^t_{il} = \begin{cases} b_{il}^t + \triangle(t, UB - b_{il}^t) & si\ a = 0 \\ b_{il}^t - \triangle(t, b_{il}^t - LB) & si\ a = 1 \end{cases}$$

  donde UB y LB son los límites superior e inferior respectivamente del dominio de b_{il}^t, la función $\triangle(t, y)$ devuelve un valor en el rango $[0, y]$ de forma que la probabilidad de que éste sea cercano a cero aumente conforme el algoritmo avanza (en [Michalewicz, 1992] se toma $\triangle(t, y) = y \cdot \left(1 - r^{(1-\frac{1}{T})^b}\right)$, donde r es un número aleatorio en el intervalo $[0, 1]$, T es el número máximo de generaciones y $b = 5$ determina el grado de dependencia con respecto a la generación actual t), y a es un valor booleano aleatorio.

  La mutación no uniforme ha sido definida también para representaciones binarias en [Michalewicz, 1992].

- *Mutación por intercambio.* Se intercambian dos elementos elegidos aleatoriamente de una permutación (véase la Sección 11.6).

11.4.4 Manejo de restricciones

El manejo de restricciones no triviales no es fácil de implementar en CE. Básicamente, hay dos enfoques para manejar individuos no factibles, dependiendo de que se permita o no la presencia de individuos no factibles en la población.

Entre los que no permiten individuos no factibles en la población, podemos citar los *métodos abortivos*, que simplemente eliminan los individuos ilegales que se generen, y los *métodos anticonceptivos*, que no permiten la generación de individuos ilegales. Así pues, en determinados problemas, como son el problema del transporte o el problema del viajante de comercio [Michalewicz, 1992], el manejo de restricciones se puede llevar a cabo incorporando el conocimiento específico del problema a la representación, y diseñando métodos de inicialización de soluciones y operadores de variación que exploten tal conocimiento para garantizar que las soluciones generadas satisfacen las restricciones. Los *algoritmos de reparo* pueden usarse para transformar las soluciones no factibles en factibles, y son normalmente complejos y muy dependientes del problema.

En el otro grupo de técnicas, las que permiten la presencia de individuos no factibles en la población, podemos citar como más importantes a las *funciones de penalización*. Las técnicas de penalización [Goldberg, 1989a] consisten básicamente en degradar o penalizar la adecuación de las soluciones en relación al grado de violación de las restricciones. Así, dado el siguiente problema general de optimimización con restricciones:

$$\begin{array}{ll} Minimizar & f(\vec{x}) \\ sujeto\ a: & g_i(\vec{x}) \leq 0,\ i = 1, \ldots, n \end{array} \tag{11.1}$$

donde $\vec{x}$ un vector de m variables, lo podemos transformar al siguiente problema no restringido:

$$Minimizar \quad f(\vec{x}) + r \cdot \sum_{i=1}^{n} \Phi[g_i(\vec{x})]$$

donde Φ es la función de penalización y r es el coeficiente de penalización. Para la función de penalización existen muchas alternativas, siendo una de las más usadas $\Phi[g_i(\vec{x})] = g_i^2(\vec{x})$. El coeficiente de penalización r normalmente se fija de forma separada para cada tipo de restricción. Esta técnica trabaja bien excepto para problemas altamente restringidos, en los cuales, encontrar una solución factible es tan difícil como encontrar una solución óptima. Encontrar coeficientes de penalización adecuados para un problema concreto en CE suele ser una tarea difícil y dependiente del problema. Si se usan penalizaciones muy altas, el AE puede perder la mayoría de su tiempo evaluando soluciones ilegales, y además, puede ocurrir que para encontrar una buena solución se tenga que pasar por soluciones ilegales como intermediarias las cuales han sido altamente penalizadas y por tanto eliminadas de la población. Si se usan violaciones moderadas, el sistema puede tener mejor consideración con soluciones ilegales que con otras que no lo son. Esto hace [Michalewicz, 1992] que los AE, considerados como métodos de búsqueda débiles, se conviertan paradógicamente en métodos fuertes (dependientes del problema) en presencia de restricciones no triviales.

A continuación describimos una regla heurística que se ha mostrado muy eficiente en un gran número de problemas test con restricciones como en la Expresión 11.1, permitiendo la existencia de individuos factibles y no factibles en la población [Jiménez y Verdegay, 1999]:

- Un individuo factible $\vec{x}$ es mejor que un individuo factible $\vec{x}'$ si:

$$f(\vec{x}) < f(\vec{x}')$$

- Un individuo factible es mejor que un individuo no factible.

- Un individuo no factible $\vec{x}$ es mejor que un individuo no factible $\vec{x}'$ si:

$$\max_i\{g_i(\vec{x})\} < \max_i\{g_i(\vec{x}')\}$$

La ventajas de esta heurística radican en que no es dependiente del problema, e integra los conceptos de optimalidad y factibilidad en un mismo proceso evolutivo, en el cual, los individuos no factibles evolucionan hacia la factibilidad, y los individuos factibles evolucionan hacia la optimalidad. Además, la heurística resulta fácil de implantar con algunos esquemas básicos de selección, muestreo o sustitución generacional. Así pues, en un esquema de selección por ranking, los individuos factibles precederían a los individuos no factibles en el ranking establecido, y dentro de cada grupo los individuos estarían ordenados por las funciones $f(\vec{x})$ y $\max_i\{g_i(\vec{x})\}$ respectivamente. En una selección por torneo, el ganador del grupo se obtiene directamente con la aplicación de la regla heurística. La regla heurística se caracteriza también por ser generalizable en un entorno de optimización multi-objetivo, como veremos en el siguiente apartado.

11.4.5 Optimización Multi-objetivo

La mayor parte de los problemas de optimización del mundo real son naturalmente multiobjetivo. Esto es, suelen tener dos o más funciones objetivo que deben satisfacerse simultáneamente y que posiblemente están en conflicto entre sí. Sin embargo, con la finalidad de simplificar su solución, muchos de estos problemas tienden a modelarse como mono-objetivo. De esta forma, con la ayuda de algún conocimiento del problema, los problemas multi-objetivo se transforman usando sólo una de las funciones originales y manejando las adicionales como restricciones, o reduciendo el vector de objetivos a uno sólo el cual optimizar. Con estos enfoques se obtiene una única solución la cual que representa el mejor compromiso entre los objetivos para un ambiente de decisión concreto. Esto en muchas situaciones prácticas es un inconveniente, ya que ante situaciones diferentes del problema se requieren nuevas ejecuciones para la obtención de nuevas soluciones. El hecho de que los AE trabajen con poblaciones de soluciones, va a permitir implementaciones de éstos que permitan la obtención de un conjunto de múltiples soluciones con una sola ejecución del algoritmo, de forma que un decisor pueda elegir, a posteriori, la solución más apropiada de acuerdo al estado de decisión existente, sin necesidad de nuevas ejecuciones.

11.4.5.1 Formulación matemática

Consideraremos la siguiente formulación para un problema de optimización multi-objetivo con restricciones:

$$\begin{array}{lll} Minimizar & f_i(\vec{x}), & i = 1, \ldots, n \\ sujeto\ a: & g_j(\vec{x}) \leq 0, & j = 1, \ldots, m \end{array} \tag{11.2}$$

donde $\vec{x} = (x_1, \ldots, x_p)$ es un vector de parámetros reales $x_k \in \Re$ pertenecientes a un dominio $[l_k, u_k]$, $k = 1, \ldots, p$, y $f_i(\vec{x})$, $g_j(\vec{x})$ son funciones arbitrarias. Llamaremos $\mathcal{F}$ al *espacio de las soluciones factibles* o *región factible*, el cual está formado por los vectores $\vec{x}$ que satisfacen las restricciones $g_j(\vec{x}) \leq 0$. $\mathcal{S}$ es el *espacio de búsqueda completo* formado por todos los posibles vectores $\vec{x}$. El espacio de búsqueda $\mathcal{S}$ se define usualmente como un rectángulo n-dimensional en $\Re^p$. Es claro que $\mathcal{F} \subseteq \mathcal{S}$.

11.4.5.2 Pareto Optimalidad

El concepto de Pareto optimalidad fué formulado por Vilfredo Pareto en el siglo XIX y constituye por sí mismo el origen de la optimización multi-objetivo.

Una solución factible $\vec{x} \in \mathcal{F}$ para el problema representado por la Expresión 11.2 *domina* a otra solución $\vec{x}' \in \mathcal{F}$ si:

$$\begin{array}{rll} & f_i(\vec{x}) \leq f_i(\vec{x}') & para\ todo\ i = 1, \ldots, n \\ y & f_i(\vec{x}) < f_i(\vec{x}') & para\ al\ menos\ un\ i = 1, \ldots, n \end{array}$$

Una solución factible $\vec{x} \in \mathcal{F}$ es Pareto óptima o *no dominada* para el problema planteado si no existe ninguna solución $\vec{x}' \in \mathcal{F}$ tal que $\vec{x}'$ domine a $\vec{x}$. Dicho de otra forma, una solución es Pareto óptima si no existe ninguna otra solución que mejore en algún objetivo sin empeorar simultáneamente otro objetivo.

El conjunto de soluciones del problema, denominado *conjunto Pareto optimal* $\mathcal{P}$ está formado por el conjunto de soluciones Pareto óptimas para el problema.

En el espacio de los objetivos, las soluciones no dominadas para el problema planteado son conocidas como el *frente Pareto optimal* o simplemente *frente Pareto* $\mathcal{PF}$ definido como:

$$\mathcal{PF} = \left\{ \vec{f}(\vec{x}) \,|\vec{x} \in \mathcal{P} \right\}$$

11.4.5.3 Aspectos relevantes en Computación Evolutiva Multi-objetivo

Un aspecto importante en CE Multi-objetivo es el la *diversidad*. Las técnicas de diversidad en optimización evolutiva multiobjetivo fueron originalmente sugeridas en [Goldberg, 1989a] a finales de los ochenta y su importancia radica fundamentalmente en dos aspectos. Primero, las múltiples soluciones capturadas mediante el algoritmo deben cubrir el frente Pareto que constituye la solución del problema. Esto significa que el algoritmo debe buscar soluciones no dominadas de una forma diversificada. Segundo, si todos los individuos no dominados de la población tienen la misma probabilidad de selección, es decir, son igual de buenos, esto puede provocar el fenómeno del *genetic drift*, el cual causa que la población converja a sólo una pequeña región del

espacio de solución. La técnica de diversidad debe entonces poder discriminar entre dos individuos no dominados, de forma que el individuo no dominado que produzca mejor diversidad tenga mayor probabilidad de selección.

En el diseño de AE multi-objetivo, el concepto Pareto, el manejo de restricciones y la técnica de diversidad deben integrarse junto con el resto de componentes que caracterizan a los AE. La selección por ranking y el torneo binario son los esquemas usados en la literatura, y se ha reconocido ampliamente la importancia del uso de la estrategia elitista. A raiz de la primera propuesta de Goldberg fueron apareciendo multitud de algoritmos multi-objetivo basados en el concepto Pareto que incluían técnicas explícitas de diversidad, dando así lugar a la 1^a *generación* de algoritmos evolutivos multi-objetivo, donde MOGA, NSGA y NPGA fueron los algoritmos más reconocidos. La inclusión del elitismo en los algoritmos evolutivos multi-objetivo dió lugar a la 2^a *generación* y última de este tipo de algoritmos, cabiendo citar a NSGA-II, DPGA, SPEA, SPEA2, PESA, PESA-II, PAES, MOMGA o MμGA [Coello y otros, 2002] como los más representativos.

Finalmente, es de extremada importancia para la evaluación de estos algoritmos la disponibilidad de un conjunto de problemas test adecuado. Recientemente, Deb et al. [Deb y otros, 2001] proponen un generador de problemas test para optimización multi-objetivo con restricciones, el cual puede ser configurado para obtener grados de dificultad deseados mediante la elección de diferentes parámetros. La Figura 11.4 muestra un conjunto de estos problemas test, y la Tabla 11.2 los parámetros para la configuración de cada uno de ellos.

$$
\begin{array}{ll}
\textit{Minimizar} & f_1(\vec{x}) = x_1 \\
& f_2(\vec{x}) = g(\vec{x})\left(1 - \sqrt{\frac{f_1(\vec{x})}{g(\vec{x})}}\right) \\
\textit{sujeto a}: & \\
& \cos(\theta)\,(f_2(\vec{x}) - e) - \sin(\theta)\, f_1(\vec{x}) \geq \\
& a\,|\sin\{b\pi\,[\sin(\theta)\,(f_2(\vec{x}) - e) + \cos(\theta)\, f_1(\vec{x})]^c\}|^d \\
& 0 \leq x_1 \leq 1 \\
& -5 \leq x_2, x_3, x_4 \leq 5
\end{array}
$$

Figura 11.4: Problemas test CTP2,...,CTP7.

[ht]

Una revisón completa de CE en optimización multiobjetivo puede encontrarse en [Coello y otros, 2002] y [Deb, 2001]. A continuación describimos el algoritmo NSGA-II, perteneciente a la 2^a generación, el cual presenta actualmente un gran impacto en la comunidad científica.

11.4.5.4 NSGA-II

En [Deb, 2001] se propone NSGA-II, un AG de ordenación no dominada con elitismo. NSGA-II utiliza una estrategia elitista junto con un mecanismo explícito de

Problema	θ	a	b	c	d	e
$CTP2$	$-0{,}2\pi$	0,2	10	1	6	1
$CTP3$	$-0{,}2\pi$	0,1	10	1	0,5	1
$CTP4$	$-0{,}2\pi$	0,75	10	1	0,5	1
$CTP5$	$-0{,}2\pi$	0,75	10	2	0,5	1
$CTP6$	$0{,}1\pi$	40	0,5	1	2	-2
$CTP7$	$-0{,}05\pi$	40	5	1	6	0

Tabla 11.2: Parámetros para los problemas test CTP2, . . . ,CTP7.

diversidad. En realidad, este algoritmo se diferencia en muchos aspectos del original NSGA, aunque los autores han decidido mantener el nombre NSGA-II para referenciar su origen.

En NSGA-II, a partir de una población P_t se crea una nueva población de descendientes Q_t aplicando *selección por torneo binario por frentes y nicho*, y los operadores evolutivos de cruce y mutación. Estas dos poblaciones se mezclan para formar una nueva población R_t de tamaño $2N$ (siendo N el tamaño de la población original P_t). La nueva población P_{t+1} se obtiene a partir de la población R_t, de la cual sobreviven los N mejores individuos tras aplicar la (*ordenación por frentes y nicho*).

Dada una población inicial aleatoria P_0, NSGA-II realiza los siguientes pasos:

1. Crear una nueva población Q_t de N individuos a partir de la población P_t (de N individuos) aplicando *selección por torneo binario por frentes y nicho*, y los operadores evolutivos de cruce y mutación,

2. Combinar las poblaciones P_t y Q_t para crear una nueva población $R_t = P_t \cup Q_t$.

3. Crear una nueva población P_{t+1} seleccionando N individuos de la población R_t utilizando la *ordenación por frentes y nichos.*

La selección por torneo binario por frentes y nicho compara dos soluciones y devuelve aquella mejor según el *operador de no dominancia y nicho.* De igual forma, la ordenación por frentes y nicho ordena los individuos según las condiciones establecidas por el *operador de no dominancia y nicho.*

Operador de no dominancia y nicho Cada individuo i tiene dos atributos:

1. Un rango de no dominancia r_i dentro de la población.

2. Una distancia de nicho local d_i, que es una medida del espacio de búsqueda alrededor de la solución i que no está ocupado por ninguna otra solución de la población.

Un individuo i se prefiere a un individuo j si ocurre alguna de las siguientes condiciones:

1. Si el individuo i tiene mejor rango que j, es decir, si $r_i < r_j$.
2. Si los individuos i y j tienen el mismo rango, pero el individuo i tiene mejor *distancia de nicho* que el individuo j, es decir, $r_i = r_j$ y $d_i > d_j$.

La primera condición asegura que la solución escogida se encuentra en un frente no dominado mejor. La segunda condición resuelve el conflicto en caso de que ambas soluciones se encuentren en el mismo frente, decidiéndose por aquella solución que tenga una mejor distancia de nicho, asegurando por tanto la diversidad.

Distancia de nicho La distancia de nicho d_i se calcula en NSGA-II de la siguiente forma:

- Sea $\mathcal{F}_i$ el conjunto de individuos de la población que pertenecen al mismo frente Pareto que el individuo i.
- Para cada función objetivo $j = 1, \ldots, m$:
  - Ordenar los individuos de $\mathcal{F}_i$ según el valor de la función objetivo f_j en una lista l_j^i.
  - f_j^{i+1} es el valor de la función j para el individuo en la lista l_j^i siguiente al individuo i.
  - f_j^{i-1} es el valor de la función j para el individuo en la lista l_j^i anterior al individuo i.
- Calcular el valor de la distancia d_i de la forma:

$$d_i = \sum_{j=1}^{m} \frac{f_j^{i+1} - f_j^{i-1}}{f_j^{max} - f_j^{min}}$$

  donde f_j^{max} y f_j^{min} son los valores máximos y mínimos para la función j en toda la población.

Esta métrica denota el valor normalizado de la mitad del perímetro del hipercubo definido por las soluciones más cercanas situadas como vértices.

La Figura 11.5 muestra los resultados obtenidos con NSGA-II para el problema test CTP2.

11.4.5.5 Métricas de evaluación

Para evaluar los algoritmos evolutivos multi-objetivo, en [Zitzler y otros, 2003] se propone el uso de la métrica ν que calcula la fracción del espacio que no es dominado por ninguna de las soluciones obtenidas por los algoritmos. El objetivo es minimizar el valor de ν. Esta métrica estima tanto la distancia de las soluciones al frente Pareto real como la diversidad de las mismas. El valor de ν puede calcularse como se muestra en la Expresión 11.3, donde P' está compuesta por las N' soluciones no dominadas

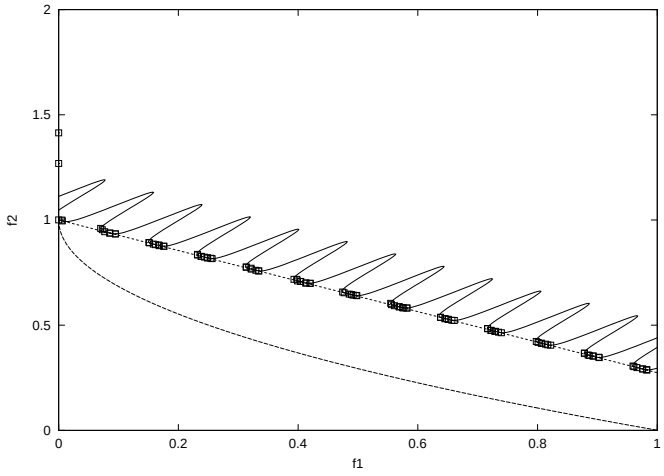

Figura 11.5: Soluciones no dominadas obtenidas con NSGA-II para el problema test CTP2.

de P, y f_j^{umax}, f_j^{umin} son los valores máximos y mínimos utópicos para el objetivo j-ésimo respectivamente.

$$\nu = 1 - \frac{\sum_{i=1}^{N'} \left[\left(f_n^{umax} - f_n^i \right) \prod_{j=1}^{n-1} \left(f_j^{sup_j^i} - f_j^i \right) \right]}{\prod_{j=1}^{n} \left(f_j^{umax} - f_j^{umin} \right)} \tag{11.3}$$

Con el objetivo de realizar una evaluación más exacta de los algoritmos evolutivos multi-objetivos, se han propuesto muchas otras métricas tanto para evaluar la convergencia como la diversidad. En [Deb, 2001] se recoge un amplio conjunto de dichas métricas. En esta sección mostramos dos de ellas.

La primera métrica, la distancia generacional Υ, evalúa la proximidad de la población al frente Pareto óptimo calculando la distancia media de la población a una población ideal P^* compuesta por N^* soluciones distribuidas uniformemente en el frente Pareto. Esta métrica se muestra en la Expresión 11.4.

$$\Upsilon = \frac{\left(\sum_{i=1}^{N'} d_i^v \right)^{1/v}}{N'} \tag{11.4}$$

En el caso más simple, $v = 1$. El parámetro d_i es la distancia Euclídea (en el espacio objetivo) entre la solución i y la solución más próxima en P^*:

$$d_i = \min_{k=1}^{N^*} \sqrt{\sum_{j=1}^{n} \left({f_j}^i - f_j^{*k} \right)^2}$$

donde f_j^{*k} es el valor del objetivo j-ésimo de la solución k-ésima en P^*.

La segunda métrica utilizada es la dispersión Δ para evaluar la diversidad de la población. La Expresión 11.5 muestra esta métrica.

$$\Delta = \frac{\sum_{j=1}^{n} d_j^e + \sum_{i=1}^{N} \left| d_i - \overline{d} \right|}{\sum_{j=1}^{n} d_j^e + N\overline{d}} \tag{11.5}$$

donde d_i puede ser cualquier medida de la distancia entre soluciones adyacentes, por ejemplo, la distancia Euclídea, y $\overline{d}$ es el valor medio de dicha medida. El parámetro d_j^e es la distancia entre las soluciones extremas en P^* y P correspondientes al objetivo j-ésimo.

11.4.6 Evaluación y validación de Algoritmos Evolutivos

Podemos tener en cuenta las siguientes consideraciones generales a la hora de evaluar y validar un AE:

- Obtener los valores mejor, medio, peor y varianza de sobre al menos 100 ejecuciones del algoritmo.
- Para la comparación con otras técnicas evolutivas, se considerarán los resultados para un mismo número de evaluaciones de la función objetivo, e iguales valores de los parámetros (tamaño de la población y probabilidades de aplicación de los operadores de variación).
- Usar un conjunto suficientemente amplio de problemas test, variando, si fuese posible, el grado de dificultad, así como el tamaño del problema (escalabilidad).
- Probar esquemas alternativos de seleccion, muestreo, sustitucion generacional y operadores de variación.
- Realizar un proceso previo de experimentación para la elección de los parámetros. Para ello se requerirán múltiples ejecuciones del algoritmo para cubrir las distintas combinaciones de los valores de los parámetros, fijando finalmente éstos a aquellos que hayan producido mejores resultados en el proceso de experimentación.
- Calcular el número de violaciones y su cuantía en optimización con restricciones.
- Usar métricas de diversidad y optimalidad en optimización multiobjetivo.

- Realizar análisis estadísticos de los resultados.
- Comparar los resultados con otras técnicas existentes, tanto deterministas como probabilistas.

11.5 Resumen

El capítulo hace un repaso a las principales técnicas que intervienen en el diseño de algoritmos evolutivos. Tras una breve introducción histórica a la Computación Evolutiva, y una clasificación y caracterización, se describe el algoritmo genético simple que D. Goldberg implementó en sus primeros trabajos, y que supuso el comienzo de esta importante disciplina. Las técnicas más usadas en la literatura para representación, selección, muestreo y sustitución generacional, así como los operadores de variación, son descritas de forma simple de cara a su utilización por parte del lector en el diseño de nuevos algoritmos evolutivos. Aspectos importantes que surgen en optimización, tales como el manejo de restricciones y la optimización multiobjetivo, son tratados también desde el punto de vista de su integración en el diseño de algoritmos evolutivos. Dos ejemplos de algoritmos evolutivos que usan representaciones ad hoc son descritos, uno para el conocido problema del viajante de comercio, y otro para el problema de la identificación de variables que surge en Minería de Datos. Finalmente, se propone un interesante problema de búsqueda, el Sudoku, que permitirá al lector comprobar su capacidad en el diseño de algoritmos evolutivos mediante el uso de las técnicas aprendidas.

11.6 Lecturas recomendadas

El libro [Goldberg, 1989a] sigue siendo una referencia en investigaciones recientes. Es un libro que presenta un carácter tanto divulgativo como de iniciación a la investigación, si bien puede resultar en determinados aspectos algo "duro" de leer y asimilar. El lector debe ser consciente de que algunos de los conceptos que ahí aparecen han sido objeto de numerosos e intensivos estudios en los años posteriores, por lo que deberán ser tenidos en cuenta sólo con carácter introductorio.

Un libro más ameno de leer y estudiar es [Michalewicz, 1992], el cual presenta también un carácter tanto docente como investigador. Este libro se centra más en las aplicaciones y en el diseño de algoritmos mediante el uso de representaciones alternativas específicas del problema. No es un libro tan técnico como el anterior, y resulta muy apropiado como revisión de los trabajos realizados hasta el momento de su publicación. Existen ediciones posteriores a 1992 más actualizadas.

Los libros [Coello y otros, 2002] y [Deb, 2001] son excelentes libros de introducción a la optimización evolutiva multi-objetivo y al manejo de restricciones, representando asímismo una referencia obligada para los investigadores en el campo. No sólo se introducen los conceptos fundamentales que giran alrededor de la optimización evolutiva multi-objetivo con restricciones, sino también repasan las técnicas evolutivas que más impacto han tenido en el campo por su eficiencia y aplicabilidad, incluyendo un

exhaustivo análisis comparativo. El prestigio de los autores avala la calidad de estos libros.

Ejercicios Resueltos

11.1. El Problema del Viajante de Comercio. El *problema del viajante de comercio*, también conocido por sus siglas en inglés como TSP, es uno de los problemas más famosos y mejor estudiados en el campo de la optimización combinatoria computacional. A pesar de la aparente sencillez de su planteamiento, el TSP es uno de los más complejos de resolver y existen demostraciones que equiparan la complejidad de su solución a la de otros problemas aparentemente mucho más complejos. El TSP está entre los problemas denominados NP-completos, esto es, los problemas que no se pueden resolver en tiempo polinomial en función del tamaño de la entrada. Sin embargo, algunos casos concretos del problema sí han sido resueltos hasta su optimización, lo que le convierte en un excelente banco de pruebas para algoritmos de optimización.

Dadas n ciudades de un territorio, y la distancia entre cada ciudad, el problema consiste en encontrar una ruta que, comenzando y terminando en una ciudad concreta, pase una sola vez por cada una de las ciudades y minimice la distancia recorrida por el viajante. Es decir, encontrar una permutación $P = \{c_0, c_1, \ldots, c_{n-1}\}$ que minimize la función:

$$d_P = \sum_{i=0}^{n-1} d(c_i, c_{(i+1) \ mod \ n})$$

donde $d(x, y)$ es la distancia entre la ciudad x y la ciudad y.

Una formulación equivalente en términos de la teoría de grafos es la de encontrar, en un grafo completamente conexo y con arcos ponderados, el ciclo hamiltoniano de menor coste. En esta formulación cada vértice del grafo representa una ciudad, cada arco representa una carretera y el peso asociado a cada arco representa la longitud de la carretera.

Dada la explosión combinatoria de las posibles soluciones, los algoritmos clásicos no son capaces de resolver el problema general. Por ello, a su solución se han aplicado distintas técnicas computacionales, normalmente basadas en heurísticas, tales como los AE. Sin embargo, desde el punto de vista teórico, estas aproximaciones no suponen una resolución real del TSP, pero ofrecen soluciones aproximadas suficientemente aceptables.

Quizás la forma más natural de representar una solución en un AE para el TSP [Grefenstette, 1987] sea la representación mediante una ruta. De esta forma, un individuo:

$$(3\ 1\ 4\ 6\ 9\ 8\ 5\ 2\ 7)$$

representa la solución que, partiendo de una ciudad origen, realiza el camino $3-1-4-6-9-8-5-2-7$, para retornar finalmente a la ciudad de origen.

La representación de las soluciones es por tanto una permutación de números enteros que representan ciudades, en la cual cada entero aparece una sola vez. Es fácil diseñar un algoritmo de inicialización aleatoria de soluciones mediante este tipo de representación.

Establecida la representación de soluciones y su inicialización, el siguiente paso es diseñar operadores de variación que se adapten a la representación. Se han descrito multitud de operadores de cruce, mutación y otros, siendo los más usados el cruce PMX, la mutación por intercambio recíproco, y la inversión.

Con el *cruce PMX*, dados dos padres seleccionados para la operación de cruce, se establece una subsecuencia mediante la elección de dos puntos de cruce, y los hijos resultantes contienen las subsecuencias intercambiadas de los padres, y el resto de elementos son cambiados, mediante un algoritmo de reparo, de forma que se preserve, por un lado, el máximo de información (ciudad y orden) de los padres, y por otro, las restricciones impuestas por la representación (cada ciudad debe aparacer una sola vez).

Dados dos padres y los puntos de cruce:

$$p_1 = (1\ 2\ 3 \mid 4\ 5\ 6\ 7 \mid 8\ 9)$$
$$p_2 = (4\ 5\ 2 \mid 1\ 8\ 7\ 6 \mid 9\ 3)$$

se producen dos hijos, en principio ilegales, de la siguiente forma:

$$h_1 = (1\ 2\ 3 \mid 1\ 8\ 7\ 6 \mid 8\ 9)$$
$$h_2 = (4\ 5\ 2 \mid 4\ 5\ 6\ 7 \mid 9\ 3)$$

El algoritmo de reparo comienza estableciendo los siguientes mapeos:

$$1 \leftrightarrow 4,\ 8 \leftrightarrow 5,\ 7 \leftrightarrow 6,\ 6 \leftrightarrow 7$$

Las ciudades que presentan un conflicto son entonces cambiadas siguiendo la tabla de mapeos. Así, la ciudades 1 y 8 en h_1 son cambiadas por las ciudades 4 y 5 repectivamente de acuerdo al mapeo establecido. De igual forma, las ciudades 4 y 5 en h_2 son cambiadas por la 1 y la 8 repectivamente. Los descendientes resultantes son:

$$h_1 = (4\ 2\ 3 \mid 1\ 8\ 7\ 6 \mid 5\ 9)$$
$$h_2 = (1\ 8\ 2 \mid 4\ 5\ 6\ 7 \mid 9\ 3)$$

los cuales explotan importantes similaridades en el valor y orden simultáneamente de los padres.

Con la *mutación por intercambio recíproco*, se eligen dos ciudades aleatoriamente, y se intercambian, asegurando así una solución legal:

$$h_1 = (4\ \mathbf{2}\ 3\ 1\ 8\ 7\ 6\ 5\ \mathbf{9})$$
$$\downarrow$$
$$h_1' = (4\ \mathbf{9}\ 3\ 1\ 8\ 7\ 6\ 5\ \mathbf{2})$$

Otro operador unario usado bajo esta representación es la *inversión*, si bien su eficacia en el proceso evolutivo ha sido más cuestionada. La inversión simple establece dos puntos aleatorios entre los cuales el orden de las ciudades es invertido, garantizando una solución válida:

$$h_1' = (4\ 9 \mid 3\ 1\ 8\ 7\ 6 \mid 5\ 2)$$
$$\downarrow$$
$$h_1'' = (4\ 9 \mid 6\ 7\ 8\ 1\ 3 \mid 5\ 2)$$

11.2. Selección de variables en Minería de Datos. La propuesta de Zadeh de modelar el mecanismo del pensamiento humano con valores lingüísticos difusos en vez de con números, condujo a la introducción de la borrosidad dentro de la teoría de sistemas y al desarrollo de una nueva clase de sistemas llamados sistemas difusos. En el modelado difuso, el problema más importante es la identificación de un modelo difuso usando datos del tipo entrada-salida. De un modo similar, la *Minería de Datos* (MD) tiene como objetivo la criba de datos para revelar información útil por medio de una representación adecuada al usuario, comprimiendo enormes registros de datos. En este contexto, un problema común a ambas técnicas es cómo tratar con mecanismos de selección de variables a partir de la posible colección de variables que pueden ser consideradas de los datos disponibles.

Aunque hemos usado el término MD, utilizamos el término *Descubrimiento de Conocimiento en Bases de Datos* (DCBD) o sólo *Descubrimiento de Conocimiento* (DC) para denotar el proceso completo de extraer conocimiento de alto nivel a partir de datos de bajo nivel, y el término MD como el acto concreto de extraer patrones o modelos de los datos. Por tanto, muchos pasos preceden al de MD, y uno de ellos es el preprocesamiento de los datos para seleccionar las variables adecuadas para ser usadas en la construcción o extracción del modelo. Este paso es similar a la identificación de variables dentro del paso de identificación de la estructura, en el proceso de *Modelización Difusa*.

Dado un conjunto de datos para el que presuminos alguna dependencia funcional, surge la cuestión de si hay alguna metodología adecuada para derivar reglas (difusas) a partir de los datos que caracterizan la función desconocida, de forma tan precisa como sea posible. Recientemente se han propuesto numerosas aproximaciones para generar automáticamente reglas if-then a partir de datos numéricos sin expertos en el dominio. Este tipo de problema es similar al que podemos encontrar en el área de DC, y en este sentido describimos aquí una técnica *Soft Computing* que intenta tener en cuenta uno de los problemas fundamentales como es el preprocesamiento de los

datos para seleccionar las características o variables adecuadas para que el proceso de MD pueda realizarse con información más relevante.

A medida que intentamos resolver problemas del mundo real, nos damos cuenta de que son normalmente sistemas mal definidos, difíciles de modelar y con espacios de solución de gran escala. En estos casos, los modelos precisos son poco prácticos, demasiado caros o inexistentes. La información relevante disponible está normalmente en la forma de conocimiento empírico previo y datos del tipo entrada-salida representando instancias del comportamiento del sistema. Así pues, necesitamos sistemas de razonamiento aproximado capaces de manejar esa información imperfecta. Soft Computing es un término acuñado recientemente que describe el uso simbiótico de muchas disciplinas emergentes de computación que intentan manejar la información imperfecta. De acuerdo con Zadeh (1994): "... en contraste con lo tradicional, hard computing, soft computing es tolerante a la imprecisión, a la incertidumbre y a la verdad parcial". Dentro de estas técnicas tenemos laLógica Difusa, el Razonamiento Probabilístico, las Redes Neuronales y los Algoritmos Evolutivos. Durante los últimos años hemos visto un número creciente de algoritmos híbridos, en los que dos o más tecnologías Soft Computing se han integrado para mejorar el rendimiento global del algoritmo.

Como sucede en la MD, si intentamos revelar información útil de los datos, un paso fundamental es el preprocesamiento de los datos para seleccionar las variables más adecuadas. En el contexto del modelado y más concretamente de la modelización difusa, el objetivo es encontrar un conjunto de relaciones que describan el comportamiento presente en los datos por medio de una colección de patrones o reglas if-then. En este proceso, la identificación de la estructura de un sistema tiene que encontrar las variables que representan los datos de un modo más preciso, de entre una colección de posibles variables. En este contexto tenemos que seleccionar un número finito de variables entre una colección finita de posibles candidatos. Esto es un problema combinatorio. Una posible aproximación a este problema es intentar asignar cierto grado a cada variable dependiendo de su importancia en la consecución del objetivo final. Este proceso es similar a una fusión multisensor, que consiste en una combinación de diferentes fuentes de información en un formato de representación. Cuando cada variable representa características diferentes, tratamos con información complementaria. Su fusión nos permite resolver ambigüedad en la información.

En este proceso de fusión de las diferentes variables de entrada posibles, tenemos dos finalidades. Primero queremos determinar la importancia de las entradas a fusionar; segundo, queremos determinar los parámetros exactos de las funciones de agregación usadas en la fusión. Encontrar los mejores parámetros de la función de agregación es un proceso de optimización. Los AE han demostrado ser muy útiles para este proceso porque aportan una búsqueda robusta en espacios de búsqueda complejos y no quedan atrapados en mínimos locales como les sucede a las técnicas de gradiente descendente.

Una vez que las variables son identificadas, el siguiente paso en la identificación de la estructura tiene que ver con la identificación de relaciones o, en otras palabras, con el descubrimiento de los patrones o el modelo a partir de los datos, y su representación por medio de diferentes alternativas como por ejemplo reglas difusas. En este paso se

pueden usar diferentes técnicas Soft Computing en el contexto del DC.

En este apartado describimos un AE para identificación de variables en Minería de Datos y DC. Expresado en términos de programación matemática, el problema puede ser formulado como sigue:

$$\begin{aligned} &Minimizar \quad E = \sqrt{\frac{\sum_{t=1}^{n}\left(y_t - \left(\sum_{j=1}^{p} w_j x_j^{t\,f}\right)^{1/f}\right)^2}{n}} \\ &sujeto\ a: \quad \sum_{j=1}^{p} w_j = 1,\ 0 \le w_j \le 1,\ j = 1,\ldots,p \end{aligned} \tag{11.6}$$

donde n es el número de datos, p es el número de variables, y_t es el valor esperado de salida para el vector de variables de entrada $X_t = \{x_1^t, \ldots, x_p^t\}$, y $\left(\sum_{j=1}^{p} w_j x_j^{t\,f}\right)^{1/f}$ es el valor del modelo *media generalizada* con grado de borrosidad f. Este tipo de operador de agregación es similar al operador OWA (*Ordered Weighted Averaging*) introducido por Yager (1988). Los pesos w_j, $j = 1, \ldots, p$, han de ser averiguados.

Aunque pueden usarse numerosas conectivas de conjuntos difusos con la finalidad de agregación, el operador media generalizada satisface las propiedades deseables. Por lo tanto, el operador puede usarse como unión o como intersección en los casos extremos, y el ratio de compensación puede controlarse variando el grado de borrosidad f.

A continuación se describen las principales características del AE propuesto. Estas características son una representación de soluciones al problema, satisfacción de restricciones, mecanismos para crear una población inicial de soluciones, la función de evaluación, operadores de variación y parámetros usados..

Representación Un individuo W de la población se representa como una colección de p elementos $W = \{w_1, \ldots, w_p\}$, donde $w_j \in [0, 1]$, $j = 1, \ldots, p$, representa el peso asociado a la variable de entrada x_j.

Satisfacción de restricciones Todos los mecanismos para crear un nuevo individuo $W = \{w_1, \ldots, w_p\}$ en el proceso evolutivo, es decir los procedimientos de inicialización y los operadores de variación, aseguran que se satisfacen la restricción $\sum_{j=1}^{p} w_j = 1, 0 \le w_j \le 1, j = 1, \ldots, p$.

Población inicial El Algoritmo 11.3 obtiene una población completa de *tampob* individuos que satisfacen las restricciones impuestas. Consideramos dos procedimientos de inicialización. El procedimiento *pesos1* (véase el Algoritmo 11.4) genera una

colección de pesos $W = \{w_1, \ldots, w_q\}$ tales que $w_l \in [0,1]$, $l = 1, \ldots, q$, y $\sum_{l=1}^{q} w_l = val$, con $0 \leq val \leq 1$. El procedimiento *pesos2* (véase el Algoritmo 11.5) es una modificación del procedimiento pesos1 para fijar una proporción aleatoria de pesos iguales a cero. Nótese que para $q = p$ y $val = 1$, ambos procedimientos pesos1 y pesos2 generan una solución factible para el problema. Estos procedimientos se usan con igual probabilidad para obtener la población inicial.

Algoritmo 11.3 Población Inicial.

Entrada: *tamaño población tampob; número de variables p;*
Salida: *población* $POB = \{W_1 = \{w_1^1, \ldots, w_p^1\}, \ldots, W_{tampob} = \{w_1^{tampob}, \ldots, w_p^{tampob}\}\}$ *de tampob individuos tales que* $w_j^i \in [0,1]$, $j = 1, \ldots, p$, $i = 1, \ldots, tampob$, *y* $\sum_{j=1}^{p} w_j^i = 1$, $i = 1, \ldots, tampob$;

1: **para** $i = 1$ hasta *tampob* **hacer**
2: $aleat \leftarrow$ valor real aleatorio $\in [0,1]$;
3: **si** $aleat \leq 0{,}5$; **entonces**
4: $pesos1(entrada : p, 1; salida : W)$;
5: **si no**
6: $pesos2(entrada : p, 1; salida : W)$;
7: **fin si**
8: $W_i \leftarrow W$;
9: **fin para**

Algoritmo 11.4 pesos1.

Entrada: *número entero q, con* $1 \leq q \leq p$; *valor real val, con* $0 \leq val \leq 1$;
Salida: *colección* $W = \{w_1, \ldots, w_q\}$ *tal que* $w_l \in [0,1]$, $l = 1, \ldots, q$, *y* $\sum_{l=1}^{q} w_l = val$;

1: $w_l \leftarrow$ valor real aleatorio $\in [0,1]$, para $l = 1, \ldots, q$;
2: $w_l \leftarrow val \cdot w_l / \sum_{i=1}^{q} w_i$, para $l = 1, \ldots, q$;

Algoritmo 11.5 pesos2.

Entrada: *número entero q, con* $1 \leq q \leq p$; *valor real val, con* $0 \leq val \leq 1$;
Salida: *colección* $W = \{w_1, \ldots, w_q\}$ *tal que* $w_l \in [0,1]$, $l = 1, \ldots, q$, *y* $\sum_{l=1}^{q} w_l = val$;

1: Seleccionar aleatoriamente $K = \{k_1, \ldots, k_r\} \subseteq \{1, \ldots, q\}$ tal que $1 \leq r \leq q$;
2: $M \leftarrow \{1, \ldots, q\} - K$;
3: $w_l \leftarrow 0$, $l \in M$;
4: $pesos1(entrada : r, val; salida : V)$;
5: $w_{k_l} \leftarrow v_l$, $l = 1, \ldots, r$;

Función de evaluación La función de evaluación de los individuos $W_i = \{w_1^i, \ldots, w_p^i\}$, $i = 1, \ldots, tampob$, está determinada por la función objetivo del problema:

$$eval(W_i) = \sqrt{\frac{\sum_{t=1}^{n}\left(y_t - \left(\sum_{j=1}^{p} w_j^i {x_j^t}^f\right)^{1/f}\right)^2}{n}},\ i = 1, \ldots, tampob$$

Operadores de variación Hemos considerado tres operadores de mutación y un operador de cruce. El operador *mutación1*, Algoritmo 11.6, realiza un cambio mínimo, intercambiando dos elementos aleatorios de los padres, mientras que los operadores *mutación2*, Algoritmo 11.7, y *mutación3*, Algoritmo 11.8, usan los procedimientos *pesos1* y *pesos2* respectivamente para producir un cambio en una parte arbitraria de los padres. El operador *cruce_aritmético*, Algoritmo 11.9), produce dos descendientes mediante la combinación lineal convexa de los padres.

Algoritmo 11.6 mutación1.

Entrada: *padre* $W = \{w_1, \ldots, w_p\}$;
Salida: *descendiente* $W' = \{w'_1, \ldots, w'_p\}$;
1: Seleccionar aleatoriamente $K = \{k_1, k_2\} \subseteq \{1, \ldots, p\}$;
2: $M \leftarrow \{1, \ldots, p\} - K$;
3: $w'_l \leftarrow w_l$, $l \in M$;
4: $w'_{k_1} \leftarrow w_{k_2}$;
5: $w'_{k_2} \leftarrow w_{k_1}$;

Algoritmo 11.7 mutación2.

Entrada: *padre* $W = \{w_1, \ldots, w_p\}$;
Salida: *descendiente* $W' = \{w'_1, \ldots, w'_p\}$;
1: Seleccionar aleatoriamente $K = \{k_1, \ldots, k_r\} \subseteq \{1, \ldots, p\}$ tal que $1 < r \leq p$;
2: $M \leftarrow \{1, \ldots, p\} - K$;
3: $w'_l \leftarrow w_l$, $l \in M$;
4: $val \leftarrow \sum_{l=1}^{r} w_{k_l}$;
5: $pesos1(entrada : r, val; salida : V)$;
6: $w'_{k_l} \leftarrow v_l$, $l = 1, \ldots, r$;

Evaluación Para evaluar el AE hemos considerado un test estándar propuesto en [Dyckhoff y Pedrycz, 1984]. El conjunto de datos consiste en una colección de 2 variables de entrada x_1 y x_2 y una variable de salida. Para comprobar la eficacia del AE, hemos añadido a los datos de entrada 6 nuevas variables x_3, x_4, x_5, x_6, x_7 y x_8 formadas a partir las variables x_1 y x_2 añadiéndoles ruido uniforme en un 10 % (a las variables x_3 y x_4), en un 40 % (a las variables x_5 y x_6), y en un 100 % (a las variables

Algoritmo 11.8 mutación3.

Entrada: *padre* $W = \{w_1, \ldots, w_p\}$;
Salida: *descendiente* $W' = \{w'_1, \ldots, w'_p\}$;

1: Seleccionar aleatoriamente $K = \{k_1, \ldots, k_r\} \subseteq \{1, \ldots, p\}$ tal que $1 < r \leq p$;
2: Set $M = \{1, \ldots, p\} - K$;
3: $w'_l \leftarrow w_l$, $l \in M$;
4: $val \leftarrow \sum_{l=1}^{r} w_{k_l}$;
5: $pesos2(entrada : r, val; salida : V)$;
6: $w'_{k_l} \leftarrow v_l$, $l = 1, \ldots, r$;

Algoritmo 11.9 cruce aritmético.

Entrada: *padres* $W_1 = \{w_1^1, \ldots, w_p^1\}$ *y* $W_2 = \{w_1^2, \ldots, w_p^2\}$;
Salida: *descendencia* $W'_1 = \{w'^1_1, \ldots, w'^1_p\}$ *y* $W'_2 = \{w'^2_1, \ldots, w'^2_p\}$;

1: $c_1 \leftarrow$ valor real aleatorio $\in [0, 1]$;
2: $c_2 \leftarrow 1 - c_1$;
3: $w'^1_j \leftarrow c_1 \cdot w_j^1 + c_2 \cdot w_j^2$, $j = 1, \ldots, p$;
4: $w'^2_j \leftarrow c_2 \cdot w_j^1 + c_1 \cdot w_j^2$, $j = 1, \ldots, p$;

x_7 y x_8). La Tabla 11.3 muestra los pesos y fitness obtenidos (valores medios sobres 10 ejecuciones) considerando solamente las 2 variables de entrada del problema, para 3 valores diferentes de f (con el objetivo de poder comparar con los ejemplos de la literatura). La Tabla 11.4 muestra los pesos y fitness obtenidos teniendo en cuenta las 2 variables de entrada originales mas las 6 variables ficticias de ruido. Los resultados muestran como el AE propuesto detecta las variables que realmente influyen en la variable de salida, excluyendo aquellas ficticias introducidas artificialmente.

	$f = 0{,}39$	$f = 1{,}25$	$f = 2$
w_1	0.447	0.405	0.316
w_2	0.553	0.595	0.684
fitness	0.053	0.114	0.161

Tabla 11.3: Resultados considerando únicamente la muestra.

Ejercicios Propuestos

11.1. Sudoku. El Sudoku es el pasatiempo de moda en todo el mundo: trata de un rompecabezas matemático en el que tienes que rellenar los huecos que faltan. El objetivo es rellenar un cuadrado de 9×9 celdas (Figura 11.6), dividido a su vez en subcuadrículas de 3×3, con las cifras del 1 al 9 partiendo de algunos números ya dispuestos en algunos espacios. Está prohibido repetir ningún número en una misma fila, columna o subcuadrícula. ¡La solución es única, así que ármate de paciencia y mucha suerte!

	$f = 0{,}39$	$f = 1{,}25$	$f = 2$
w_1	0.420	0.406	0.318
w_2	0.486	0.518	0.682
w_3	0.024	0.0	0.0
w_4	0.015	0.076	0.0
w_5	0.002	0.0	0.0
w_6	0.012	0.0	0.0
w_7	0.019	0.0	0.0
w_8	0.022	0.0	0.0
fitness	0.052	0.113	0.161

Tabla 11.4: Resultados considerando la muestra con ruido.

5	3		4	8				
		6	5					
8			2	1		5		
	1					9	8	
		2	1	5	9	4		
	5	3					1	
	6			7	4	3		1
					1			
				3	5		7	4

Figura 11.6: Un Sudoku.

1. Proponer un AE para el Sudoku, estableciendo:
   - Una representacion de soluciones.
   - Una función de evaluación.
   - Un operador de cruce.
   - Un operador de mutación.
2. Implementar distintos AE mediante:
   - Seleccion proporcional, muestreo estocástico con reemplazamiento (rueda de ruleta), elitismo.
   - Selección proporcional, muestreo universal estocástico, elitismo.
   - Selección por ranking lineal, muestreo universal estocástico, elitismo.
   - Selección por torneo binario, elitismo.
3. Evaluar y comparar los distintos AE. Mostrar los valores del mejor, medio, peor y varianza sobre 100 ejecuciones.

Bibliografía

COELLO, C.A.; VELDHUIZEN, D.V. y LAMONT, G.B.: *Evolutionary Algorithms for Solving Multi-Objective Problems*. Kluwer Academic/Plenum publishers, New York, 2002.

DAVIS, L.: *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, New York, 1991.

DEB, K.: *Multi-Objective Optimization using Evolutionary Algorithms*. John Wiley and Sons, LTD, 2001.

DEB, K.; PRATAP, A. y MEYARIVAN, T.: «Constrained test problems for multi-objective evolutionary optimization». *Lectures Notes in Computer Science*, 2001, **1993**, pp. 284–298.

DEJONG, K.A.: *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*. Doctoral Dissertation, University of Michigan, 1975.

DYCKHOFF, H. y PEDRYCZ, W.: «Generalized means of model of compensative connectives». *Fuzzy Sets and Systems*, 1984, **14**, pp. 143–154.

FOGEL, L.J.; OWENS, A.J. y WALSH, M.J.: *Artificial Intelligence through Simulated Evolution*. Wiley, New York, 1966.

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

GOLDBERG, D.E.: «Sizing populations for serial and parallel genetic algorithms». En: California Morgan Kaufmann Inc. J.D. Schaffer (ed.), San Mateo (Ed.), *Proc. of the Third Intern. Conf. on Genetic Algorithms*, pp. 70–79, 1989b.

GREFENSTETTE, J.J.: «Incorporating problem specific knowledge into genetic algorithms». En: L. Davis (Ed.), *Genetic Algorithms and Simulated Annealing, Pitman, London*, , 1987.

HOLLAND, J.H.: *Adaptation in Natural and Artificial Systems*. The University of Michigan Press, Ann Arbor, 1975.

JIMÉNEZ, F. y VERDEGAY, J.L.: «Evolutionary techniques for constrained optimization problems». En: *Proc. of the 7th European Congress on Intelligent Techniques and Soft Computing (EUFIT'99), Aachen, Germany, Springer-Verlag*, , 1999.

KOZA, J.R.: *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. The MIT Press, Cambridge, Massachusetts, 1972.

MICHALEWICZ, Z.: *Genetic Algorithms + Data Structures = Evolution Programs*. Springer Verlag, 1992.

RECHENBERG, I.: *Evolutionary Estrategy: Optimization of Technical Systems According to the Priciples of Biological Evolution*. Frommann-Holzboog, 1973.

ZITZLER, E.; THIELE, L.; LAUMANNS, M.; FONSECA, C.M. y DA FONSECA, V. GRUNERT: «Performance Assessment of Multiobjective Optimizers: An Analysis and Review». *IEEE Transactions on Evolutionary Computation*, 2003, **7(2)**, pp. 117–132.