Una introducción a la Computación Evolutiva

Anselmo Pérez Serrada

9 de marzo de 1996

Índice General

Índice General

Planteamiento y objetivos

Bajo el término de ***Computación Evolutiva*** se engloba a una amplio conjunto de técnicas de resolución de problemas complejos basadas en la emulación de los procesos naturales de evolución. No es la primera vez que se hace uso de la emulación de procesos naturales para resolver problemas científicos y técnicos: las Redes Neuronales para la resolución de problemas de aprendizaje artificial y la Solificación Simulada para resolver problemas de optimización que se plantean en las ciencias físicas son seguramente los ejemplos más conocidos.

La principal aportación de la Computación Evolutiva a la metodología de resolución de problemas consiste en el uso de mecanismos de selección de soluciones potenciales y de construcción de nuevos candidatos por recombinación de características de otros ya presentes, de modo parecido a como ocurre en la evolución de los organismos naturales. Debe quedar bien claro que la Computación Evolutiva no trata tanto de reproducir ciertos fenómenos que suceden en la naturaleza como de aprovechar las ideas genéricas que hay detrás de ellos. Efectivamente, en el momento en que se tienen varios candidatos a solución para un problema surge inmediatamente la necesidad de establecer criterios de calidad y de selección —en función de la calidad— y también la idea de combinar características de buenas soluciones para obtener otras mejores. Dado que fue en el mundo natural donde primeramente se han planteado problemas de ese tipo, no tiene nada de extraño el que al aplicar tales ideas en la resolución de problemas científicos y técnicos se obtengan procedimientos bastante parecidos a los ya encontrados por la Naturaleza tras un largo periodo de adaptación.

Precisamente, la Computación Evolutiva 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*, que con el paso del tiempo ha sido considerado como el punto de arranque de la Computación Evolutiva. Los intereses de Holland y sus colaboradores fueron en principio académicos: esencialmente trataban 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; a la hora de llevarlas a la práctica las ideas de Holland resultaban, cuando menos, poco eficientes.

Sin embargo, a mediados de los '80 la aparición de computadores de altas prestaciones y bajo costo cambia por completo el panorama: la Computación Evolutiva se puede utilizar para resolver con éxito ciertos tipos de problemas de la ingeniería y de las ciencias sociales que anteriormente eran difícilmente. De ésta época son el libro de Goldberg [GOLD89] y el de Davis [DAVI91]; en ellos se plantean y resuelven problemas "de la vida real" en los que hay de por medio apreciables cantidades de dinero. Posiblemente como consecuencia del esfuerzo divulgador de éstos autores y debido también a las nuevas posibilidades que se vislumbraban, el desarrollo de la Computación Evolutiva a partir de entonces fue espectacular. En la actualidad funcionan con éxito aplicaciones industriales de los algoritmos evolutivos en campos tan dispares como el diseño de turbinas multietapa, el control de redes de distribución de gas, la planificación de tareas, el cálculo de estrategias de mercado o el cortado de piezas con mínimo desperdicio, por citar algunas.

Sin embargo ese desarrollo no ha estado exento de inconvenientes: por un lado se ha generado en pocos años una gran cantidad de información, extremadamente dispersa y de desigual calidad, y por otro lado los escasos requisitos de aplicabilidad así como gran atractivo conceptual de la disciplina hace que muchos autores caigan en la tentación de aplicar a la ligera —esto es, de modo inapropiado o sobre problemas inadecuados— los procedimientos de la Computación Evolutiva.

Objetivos

Es precisamente en el contexto recién señalado donde se debe enmarcar el presente documento. A mediados del año 1995 manifesté a los profesores Ricardo del Olmo Martínez y Alberto Gómez Gómez mi interés por realizar mi Proyecto Fin de Carrera sobre algún tema relacionado con los Algoritmos Genéticos, si es que ello estaba dentro de los intereses su departamento. Tras ciertas deliberaciones se llegó a la conclusión de que para poder desarrollar proyectos "útiles" desde un punto de vista ingenieril antes era imprescindible conocer exactamente de *qué* herramientas se disponía y tener criterios concretos sobre *cuándo* y *cómo* aplicarlas. Ésto, que ya está plenamente establecido en otras ramas de la ingeniería, aún se encuentra en plena fase de elaboración para la Computación Evolutiva. Por ese motivo se resolvió plantear el proyecto basándose en los siguientes objetivos:

- Realizar un estudio coherente y pormenorizado de los métodos y procedimientos propios de la Computación Evolutiva, prestando especial atención a sus posibilidades de aplicación en el ámbito empresarial.

- Realizar una búsqueda bibliográfica lo más amplia posible de toda la información relacionada con este tema, seleccionando la que sea más relevante desde un punto de vista práctico. En ésta búsqueda se incluye el *software* de dominio público disponible a través de la red INTERNET.

- Desarrollar algoritmos que puedan ser implantados directamente para resolver problemas prácticos.
- Implantar la máxima cantidad de los algoritmos anteriores sobre problemas típicos o ilustrativos de cada caso.
- Resolver completamente algún problema de "ingeniería real" sirviéndose de las rutinas implantadas.
- Contrastar las prestaciones de los Algoritmos Evolutivos con otros métodos de resolución de problemas de búsqueda y aprendizaje.
- Mostrar las posibilidades de desarrollo de la Computación Evolutiva, señalando claramente en qué tipos de problemas del ámbito empresarial resulta más conveniente su aplicación y también en cuáles no merecería la pena implantarlos.

Con la intención de cumplir esos objetivos se han elaborado los capítulos que se describen a continuación.

Planteamiento General

El presente documento consta de siete capítulos con los siguientes contenidos:

Capítulo 1: En él se presenta la Computación Evolutiva, la estructura genérica de los algoritmos que de ella se derivan (Algoritmos Evolutivos) y los principales paradigmas a que da lugar. También se describe el lugar ocupado por la Computación Evolutiva en el contexto de los problemas de búsqueda y optimización y se comparan, en términos de generalidad y eficacia, los Algoritmos Evolutivos con las técnicas clásicas de resolución de dichos problemas.

Capítulo 2: En él se describe el paradigma más importante de la computación evolutiva: los Algoritmos Genéticos, mostrando su estructura y componentes básicos así como los métodos y criterios necesarios para implantarlos. Todos ellos se ilustran mediante un ejemplo práctico. En la segunda parte del capítulo se realiza un análisis teórico de los Algoritmos Genéticos con el fin de explicar su funcionamiento interno y justificar las razones de su eficiencia añadida.

Capítulo 3: En él se enumeran las limitaciones e inconvenientes que plantea el uso de Algoritmos Genéticos, muchas de las cuales son comunes a otras técnicas evolutivas. Se explican pormenorizadamente las diversas alternativas que existen para resolver o mitigar tales inconvenientes.

Capítulo 4: En él se introducen cuatro mecanismos suplementarios con los que se se puede extender el modelo básico de los Algoritmos Genéticos, a saber, mecanismos de duración, de escisión, de paralelismo y de diploidismo.

Capítulo 5: En él se describen los Programas de Evolución, que constituyen un nuevo paradigma de la Computación Evolutiva con el que se introduce directamente conocimiento específico del problema en el interior de un Algoritmo Genético. Se muestran detenidamente varios modos de introducir dicho conocimiento específico según el tipo de problema que se aborde. Al final del capítulo se enumeran algunos problemas típicos para los que los Programas de Evolución han proporcionado soluciones aceptables con rendimientos similares o mejores que otros procedimientos.

Capítulo 6: En él se describen las Estrategias Evolutivas, que son un paradigma de la Computación Evolutiva especializado en optimizaciones paramétricas. Debido a su flexibilidad, las estrategias evolutivas pueden resolver problemas paramétricos muy generales; se ilustra esta propiedad resolviendo completamente un problema de minimización de los costes de seguimiento de procesos dinámicos, el cual es difícil de resolver eficientemente por métodos no evolutivos.

Capítulo 7: En él se muestran las principales maneras que existen para introducir la Computación Evolutiva en la resolución de problemas aprendizaje artificial. Se presta especial atención a dos formulaciones, una que hace uso de los Algoritmos Genéticos y otra que hace uso de los Programas de Evoloción. Se ilustran ambas sobre un problema industrial de aprendizaje de estrategias de control.

Se da fin al documento con una sección de conclusiones en la que, en función de los objetivos anteriormente trazados, se extraen los resultados más destacados de la exposición precedente.

Capítulo 1

Introducción a la Computación Evolutiva

La Computación Evolutiva (en lo sucesivo, CE) es un enfoque alternativo para abordar problemas complejos de *búsqueda* y *aprendizaje* a través de modelos computacionales de procesos evolutivos. Las implantaciones concretas de tales modelos se conocen como algoritmos evolutivos (en lo sucesivo, AEs). El propósito genérico de los EAs consiste en guiar una búsqueda estocástica haciendo evolucionar a un conjunto de estructuras y seleccionando de modo iterativo las más adecuadas.

La Computación Evolutiva forma parte a su vez de un conjunto de metodologías de resolución de problemas que remedan con mayor o menor exactitud procesos naturales, como las Redes Neuronales o la Solidificación Simulada. Todas ellas se engloban bajo el término *Computación Natural*.
Pese a lo que pudiera parecer, el desarrollo de la Computación Natural no se debió tanto a disquisiciones académicas como a la necesidad de resolver problemas cada vez más complejos en múltiples campos del conocimiento. Tal como se plantean dichos problemas las técnicas clásicas de búsqueda determinística y analítica no suelen ser de gran utilidad, lo que obliga a desarrollar nuevos paradigmas menos analíticos y más sintéticos. En tal circunstancia, la búsqueda de analogías con la naturaleza es más una necesidad que una preferencia estética.

Debe quedar claro en todo momento que no se persigue una *simulación* de los procesos naturales, sino más bien una *emulación* de dichos procesos. Por eso, un Algoritmo Evolutivo será tanto mejor cuanto mejores resultados proporcione en la resolución del problema planteado, independientemente de su fidelidad a la biología. De hecho, la mayoría de algoritmos que se derivan de este enfoque son exageradamente simplistas desde un punto de vista biológico, pero lo suficientemente complejos como para proporcionar robustos y potentes mecanismos de búsqueda.

1.1 Estructura genérica de los Algoritmos Evolutivos

Como ya quedó dicho, la metodología de la Computación Evolutiva se plasma en los ***Algoritmos Evolutivos***. En general, se llama Algoritmo Evolutivo (en lo sucesivo, AE) a cualquier procedimiento estocástico de búsqueda basado en el principio de evolución. Dicho principio tiene una doble vertiente: la finalidad última es la "supervivencia del más apto", el modo de conseguirlo es por "adaptación al entorno". De un modo más gráfico, los más aptos tienen más posibilidades de sobrevivir y, como resultado, más oportunidades de transmitir sus características a las generaciones siguientes.

Más concretamente, al ejecutar un AE una población de individuos, que representan a un conjunto de candidatos a soluciones de un problema, es sometida a una serie de transformaciones con las que se actualiza la búsqueda y después a un proceso de selección que favorece a los mejores individuos. Cada ciclo de *transformación+selección* constituye una *generación*. Se espera del AE que, tras cierto número de generaciones —ie., de iteraciones— el mejor individuo esté razonablemente próximo a la solución buscada.

Dicho de otra manera, la CE trata de desarrollar mecanismos estocásticos de búsqueda en paralelo con los que mejorar las técnicas clásicas de búsqueda determinista cuando estas no sean buenas o ni siquiera existan. Para que la mejora sea efectiva, tales mecanismos deben estar "dirigidos", de ahí la necesidad de introducir un procedimiento de *selección*. En definitiva, para poder emular suficientemente el proceso de evolución un EA debe disponer de:

1. Una población de posibles soluciones debidamente representadas a través de ***individuos***.

2. Un procedimiento de ***selección*** basado en la ***aptitud*** de los individuos.

3. Un procedimiento de ***transformación***, esto es, de construcción de nuevas soluciones a partir de las disponibles actualmente.

Una vez que estos elementos han sido convenientemente especificados se implanta el AE siguiendo el esquema general de la figura 1.1.

Sobre ese esquema general se han desarrollado cuatro paradigmas fundamentales:

- ***Los Algoritmos Genéticos*** [Holland]: Se hace evolucionar una población de enteros binarios sometiéndolos a transformaciones unitarias y binarias genéricas y a un proceso de selección.

```
BeginAlgorithm{EvolutionAlgorithm}
 P[0] = InitPop();                          Población inicial
 FitP[0] = EvalPop(P[0]);                   Aptitudes iniciales
 for( t = 0; t < MaxNumGen; t++) {
  Q[t] = SelectBreedersFrom(P[t]);          Selección de reproductores
  Q[t] = Transform(Q[t]);                   Reproducción
  FitQ[t] = EvalPop(Q[t]);
  P[t] = SelSurv(P[t],Q[t],FitP[t],FitQ[t]);
  FitP[t] = EvalPop(P[t]);                  Se evalúan la nueva población
  if( ChkTermCond(P[t], FitP[t]) )          Condición de terminación
   return;
 }
EndAlgorithm
```

Figura 1.1: Estructura general de un Algoritmo Evolutivo

- ***Los Programas Evolutivos*** [Michalewicz]: Se hace evolucionar una población de estructuras de datos sometiéndolas a una serie de transformaciones específicas y a un proceso de selección.

- ***Las Estrategias Evolutivas*** [Rechenberg y Schwefel]: Se hace evolucionar una población de números reales que codifican las posibles soluciones de un problema numérico y los tamaños de salto. La selección es implícita.

- ***La Programación Evolutiva*** [Fogel]: Se hace evolucionar una población de máquinas de estados finitos sometiéndolas a trasformaciones unitarias.

Todos ellos se estudiarán detenidamente en los capítulos sucesivos.

1.2 La CE en el contexto de los problemas de búsqueda y optimización

Antes de entrar con detalle en la descripción de los cuatro paradigmas de la computación evolutiva conviene hacer un breve repaso a los conceptos elementales de la teoría general de búsqueda y optimización. Así se introducirá la terminología básica y además se situarán los EAs en el marco de comparación adecuado.
Se debe notar que los términos *búsqueda* y *optimización* son dos facetas de un mismo concepto: La diferencia estriba en que al decir 'búsqueda' se quiere hacer hincapié en el proceso en sí, mientras que al decir 'optimización' se destaca más el resultado de tal proceso.

A grandes rasgos, los procedimientos de optimización tratan de resolver el problema

$$\mathbf{x}^{\circ} = \underset{\mathbf{x}\in\mathcal{X}}{\text{argopt}}\, \mathbf{f}(\mathbf{x})$$

en el que la aplicación $\mathbf{f}(\cdot)$ asocia a cada elemento $\mathbf{x}$ del conjunto $\mathcal{X}$ un vector de números reales $\mathbf{f}(\mathbf{x}) \in \Re^r$. Naturalmente, no es necesario que $\mathbf{f}(\mathbf{x})$ tenga una forma cerrada o explícita.

Los candidatos a solución $\mathbf{x}$ son estructuras (conjuntos, formaciones, árboles, grafos, etc.) de objetos los cuales se pueden caracterizar inequívocamente a través de m atributos en cierto orden. Al dominio del problema se le denota $\mathcal{X}$ y no necesariamente coincide con el *espacio de búsqueda* del algoritmo, que se denota X.

Sin pérdida de generalidad se puede admitir que se resuelve una maximización, pues para minimizar basta con cambiarle el signo a $\mathbf{f}(\mathbf{x})$. Para mayor sencillez se supondrá también que la optimización es *mono–objetivo*, esto es, que $f(\mathbf{x})$ devuelve un escalar. Las optimizaciones multiobjetivo se tratarán separadamente en el apartado 4.2.3.
Por último, la notación $\mathbf{x} \succ \mathbf{y}$ denota '$\mathbf{x}$ es mejor que $\mathbf{y}$', es decir $f(\mathbf{x}) > f(\mathbf{y})$ si se está maximizando y al revés si se está minimizando.

1.2.1 Clasificación general de los procedimientos de optimización

De modo general se pueden establecer cinco criterios de clasificación para los procedimientos de optimización:

1. Basándose en la naturaleza de las soluciones las optimizaciones pueden ser:

   - ***Numéricas***: Si la solución queda completamente especificada en términos de un conjunto de m parámetros o atributos.
   - ***Combinatorias***: Si para especificar la solución no sólo hay que especificar un conjunto de m parámetros, sino también el orden (total o parcial) con que estos se combinan para dar dicha solución.
     En último término puede ocurrir que sea irrelevante la naturaleza de los atributos y sólo importe su orden. En tal caso se habla de *problemas basados en el orden*.

2. Basándose en el grado de aleatoreidad que se le da al proceso de búsqueda las optimizaciones pueden ser:

   - ***Deterministas o dirigidas***: El procedimiento de búsqueda es completamente determinista, es decir, en las mismas condiciones de partida proporciona idénticos resultados.

- ***Aleatorias o al azar***: El procedimiento de búsqueda es completamente aleatorio. Habitualmente, se delimita una región de búsqueda y se toman puntos al azar dentro de ellas. Después mediante argumentos estadísticos se puede dar una estimación de máxima verosimilitud para el valor del óptimo.
- ***Estocásticas u orientadas***: Se combinan en proporción variable la búsqueda determinista con la búsqueda aleatoria. La componente determinista orienta la dirección de búsqueda y la aleatoria se encarga de la búsqueda local.

Las técnicas clásicas de optimización, ya sean analíticas o enumerativas, son todas deterministas; por lo común son las más eficientes, pero son muy específicas y precisan mucho conocimiento adicional sobre la función objetivo así como el uso de hipótesis suplementarias de "buen comportamiento", entre otros inconvenientes. En cuanto a las técnicas aleatorias la situación es opuesta: no requieren ninguna información adicional y se pueden aplicar a cualquier tipo de problemas, pero son poco eficientes. Por estos motivos resulta conveniente buscar un punto intermedio entre la máxima eficiencia de las técnicas deterministas y la máxima eficacia de las aleatorias.

3. Basándose en la dirección preferente de búsqueda las optimizaciones pueden ser:

- ***En profundidad*** ó ***Explotadoras***: La búsqueda da prioridad a la *explotación* de las soluciones disponibles antes que a la exploración de nuevas soluciones.
- ***En anchura*** ó ***Exploradoras***: La búsqueda da prioridad a la *exploración* de nuevas soluciones antes que a la explotación de las disponibles.

En pocas palabras, la exploración trata de obtener más conocimiento del espacio de búsqueda y la explotación trata de aprovechar el que ya se tiene. Muchos métodos de búsqueda no son explotadores o exploradores puros, sino que combinan ambos procedimientos. De hecho es deseable tener control sobre la relación explotación$\rightleftharpoons$exploración. Idealmente, esa relación debería poderse expresar como un cociente entre dos parámetros al que se llama *grado de penetración*.

4. Basándose en el número de candidatos a solución que se mantienen simultáneamente las búsquedas pueden ser

- ***Simples***: Se mantiene un sólo candidato a solución que se va actualizando sucesivamente para proporcionar —presumiblemente— soluciones cada vez más exactas del problema.

- ***Múltiples***: Se mantienen simultáneamente varios candidatos a solución con los cuales se va acotando cada vez con más precisión la región (o regiones) donde se encuentra el/los óptimo/s. Son las más apropiadas para implantaciones en paralelo.

Aunque son computacionalmente más costosas, las búsquedas múltiples presentan tal cantidad de ventajas sobre las búsquedas simples que se deberían usar siempre que fuera posible, aunque no se disponga de procesadores en paralelo.

5. Basándose en la información disponible sobre la función a optimizar las búsquedas pueden ser

- ***Ciegas***: El proceso a optimizar funciona como una *caja negra* que ante ciertos valores de los parámetros devuelve un valor del objetivo. Es decir, no se dispone de ninguna información explícita sobre la aplicación $\mathbf{f}(\mathbf{x})$. En la bibliografía inglesa a estos métodos de optimización se les designa como *blackbox optimization*, que se puede traducir como 'optimización de cajas negras'. La ventaja de estos métodos es que proporcionan algoritmos de búsqueda de *propósito general*, los cuales son muy fáciles de implantar para un problema específico.
- ***Heurísticas***: Se dispone de cierta información explícita sobre el proceso a optimizar, la cual se puede aprovechar para guiar la búsqueda. En la jerga del ramo a dicha información útil para la búsqueda se le llama *conocimiento específico*. Las técnicas heurísticas proporcionan *algoritmos dedicados* de búsqueda, esto es, específicos para un problema concreto y difícilmente adaptables para cualquier otro.

Tradicionalmente se ha tratado de añadir la máxima cantidad de conocimiento específico en los problemas de optimización, dado que es lo que más eficacia da a la búsqueda. Sin embargo, en problemas reales de mediana complejidad resulta muy difícil, cuando no imposible, encontrar tal información y la que se encuentra no suele ser de muy buena calidad. A efectos prácticos, el conocimiento específico sólo sirve verdaderamente cuando es de buena calidad, y aun así se debe tener cuidado porque acentúa la tendencia a estancar la búsqueda en óptimos locales.

La cuestión de si añadir conocimiento específico a un problema y de cuánto y cómo añadirlo es fundamental en la teoría moderna de optimización, por lo que se volverá a tratar con más detalle.

1.2.2 Complejidad de los problemas de optimización

Atendiendo a su complejidad, se dice de un problema de optimización que es *polinómicamente programable* cuando existen procedimientos de resolución que dan la respuesta tras realizar un número de operaciones que es —a lo sumo— una función polinómica del tamaño del problema. Este es un requisto mínimo de complejidad, es decir, si un problema no es polinómicamente programable todos los métodos de resolución serán inútiles para tamaños crecientes del problema ya que ningún computador sería capaz de afrontarlos en un tiempo razonable.

Por desgracia, gran número de los problemas de optimización que surgen en la práctica no son polinómicamente programables y, lo que es peor, existen buenas razones teóricas para pensar que ésta es una característica intrínseca de tales problemas, esto es, es imposible encontrar un procedimiento polinómico de búsqueda que los resuelva.

Así las cosas, y dado que se deben resolver de un modo u otro tales problemas caben las siguientes alternativas:

1. Modificar el problema de tal manera que admita un algoritmo de resolución aproximada, esto es, que encuentre buenas soluciones pero no necesariamente la mejor. A tales soluciones se les llama *cuasióptimos*.

2. Desarrollar algoritmos específicos que encuentren la solución en algunos casos, pero no necesariamente en todos, con la esperanza (más o menos fundada) de que los problemas que surgen en la práctica entren en el primer grupo.

3. Rebajar las restricciones del problema para que admita un algoritmo de resolución en tiempo polinómico y obtener varias soluciones. Se elegirá la que más se ajuste a las restricciones originales.

4. Investigar el intervalo de algoritmos 'cuasi-polinómicos'. Los algoritmos de resolución en tiempo no exponencial (por ejemplo, proporcional a $2^{\sqrt{m}}$ donde m es el tamaño del problema) dan algunas esperanzas de resolver problemas moderadamente grandes.

Los nuevos métodos de optimización deben poder contemplar la utilización de estas u otras alternativas para proporcionar soluciones razonables a problemas de este tipo. Concretamente, los AEs suelen hacer uso de la primera alternativa, ie., buscan cuasióptimos.

1.2.3 Fortaleza y robustez. El conflicto entre generalidad y eficiencia

Tradicionalmente, la característica global más importante de un método de búsqueda es su ***fortaleza***. Esta se mide en términos de eficiencia: un método es tanto más fuerte cuanto más eficientemente lleve a cabo su tarea.

Ahora bien, un postulado fundamental de la teoría de optimización afirma que la eficiencia y la generalidad son objetivos contrapuestos, esto es, cuanto más fuerte sea un método menor cantidad de problemas será capaz de resolver y viceversa. Es precisamente a esto a lo que se refiere la bibliografía al hablar del *conflicto entre generalidad y eficiencia*.

Los métodos heurísticos y/o deterministas son los que más fortaleza poseen. Tradicionalmente han sido los más usados porque es preferible tener poca generalidad antes que poca eficiencia. Sin embargo, el incremento de la cantidad y complejidad de los problemas a resolver llevó a mediados de la década de los '60 formular la siguiente cuestión:

> Admitiendo que para los procedimientos tradicionales de búsqueda la relación generalidad vs. eficiencia es inevitablemente constante ¿Existen otros procedimientos de búsqueda para los que dicho producto sea mayor?

Esto lleva a cuestionar el papel fundamental que se otorga a la fortaleza para evaluar la calidad de un método de búsqueda y a introducir una nueva característica de los método de búsqueda: la ***robustez***. Un método es tanto más robusto cuanto mayor sea la relación generalidad×eficiencia. Desde esta perspectiva, se trata de determinar si existen métodos de búsqueda más robustos que los tradicionales.

Como consecuencia del gran número de investigaciones iniciadas en aquella época se han encontrado efectivamente nuevos procedimientos de búsqueda más robustos que las tradicionales. Para ello ha sido necesario relativizar las exigencias de fortaleza y aprovechar las buenas cualidades en términos de robustez que proporcionan el no determinismo y la *opacidad*, ie., la no utilización de conocimiento específico.

Todo esto se expresa gráficamente en la figura 1.2. Las formas hiperbólicas de las regiones muestran la tendencia a mantener constante la relación generalidad×eficiencia. La Computación Evolutiva trata de cubrir la zona superior, de mayor robustez que la ocupada por las técnicas clásicas.

1.3 Técnicas clásicas de búsqueda y optimización

No es el objetivo de esta memoria hacer un estudio comparativo de las diversas técnicas de búsqueda y optimización; por ello se hará simplemente un recorrido somero sobre las técnicas de optimización más comunes, no tanto para describirlas como para enumerar sus limitaciones en términos de robustez. Las descripciones detalladas de cada técnica así como sus múltiples variantes pueden encontrarse en la bibliografía.

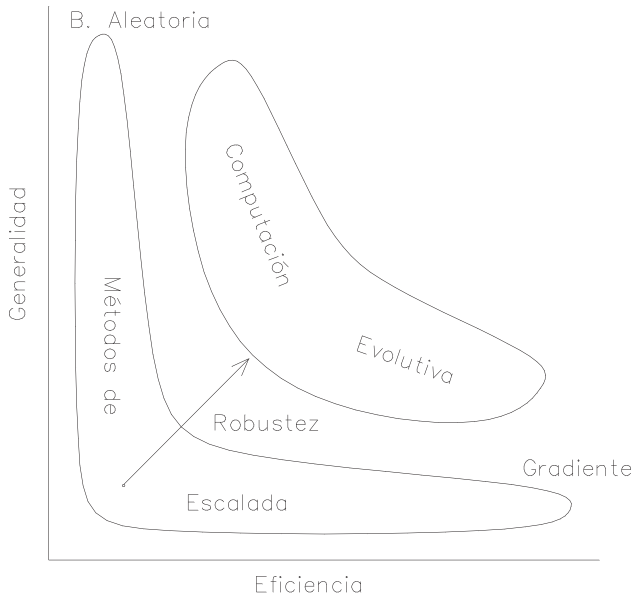

Figura 1.2: El conflicto generalidad×eficiencia

Técnicas analíticas de búsqueda y optimización

De manera general, las técnicas clásicas de búsqueda y optimización se pueden clasificar como analíticas o como enumerativas. Las *técnicas analíticas* están especialmente indicadas para problemas numéricos con las siguientes propiedades:

- El espacio de búsqueda ha de estar claramente definido y ser contínuo y regular para que en todo momento tenga sentido el concepto de 'vecindad'.
- La función objetivo no debe ser ruidosa (ie., a cada entrada le ha de corresponder siempre la misma salida) y ha de ser lo suficientemente 'lisa' como para que en todo momento se pueda definir una 'dirección preferente de búsqueda'.

Un ejemplo de este tipo de problemas son los que se plantean habitualmente en programación estática. Tales problemas tienen la siguiente forma genérica:

$$\begin{cases} \text{maximizar} & f(\mathbf{x}) \\ \text{sujeto a} & \mathbf{g}(\mathbf{x}) \geq \mathbf{0} \quad \text{y} \quad \mathbf{x} \geq \mathbf{0} \qquad (\mathbf{x} \in \Re^m) \end{cases}$$

y bajo ciertas condiciones de regularidad se pueden resolver a través de las condiciones de Kuhn-Tucker

$$\begin{array}{rclcrcl} \dfrac{\mathbf{df}}{\mathbf{dx}} - \lambda \cdot \dfrac{\mathbf{dg}}{\mathbf{dx}} & \geq & \mathbf{0} & \qquad & \mathbf{x}' \cdot \left(\dfrac{\mathbf{df}}{\mathbf{dx}} - \lambda \cdot \dfrac{\mathbf{dg}}{\mathbf{dx}} \right) & = & \mathbf{0} \\ \mathbf{g}(\mathbf{x}) & \geq & \mathbf{0} & & \lambda' \cdot \mathbf{g}(\mathbf{x}) & = & \mathbf{0} \\ \mathbf{x} & \geq & \mathbf{0} & & \lambda & \geq & \mathbf{0} \end{array}$$

siendo λ un vector del mismo tamaño que $\mathbf{g}(\mathbf{x})$.

Las técnicas clásicas hallan el óptimo resolviendo de modo directo o indirecto esas ecuaciones. La *resolución directa* sólo es abordable cuando $f(\mathbf{x})$ se expresa como una composición analítica de funciones elementales y aun en tal caso puede ocurrir que no sea el mejor método. Los *métodos indirectos de resolución* buscan el óptimo trepando iterativamente la función objetivo en la dirección de máxima pendiente. Procediendo de este modo se garantiza encontrar un óptimo local o *subóptimo* —ie., un punto rodeado de puntos peores que él—, aunque no necesariamente el óptimo global.

Este procedimiento de ***escalada*** da lugar a algoritmos de resolución muy sencillos, siempre y cuando se disponga de una buena técnica de determinación de la dirección de máxima pendiente. Habitualmente, tal dirección se determina a través de modelos locales aproximados de primer o segundo orden (estrategias de Newton-Raphson) auxiliados por procedimientos diversos que evitan el estancamiento de la búsqueda en mesetas, en grietas o en puntos singulares.

En la figura 1.3 se muestra un algoritmo simple de optimización por escalada. En cada iteración se determina una región de alta pendiente en la vecindad del punto actual y se selecciona el mejor punto de dicha vecindad; ésto suele ser más eficaz que determinar directamente la dirección de máxima pendiente. Si el nuevo punto proporciona un valor de la función objetivo estrictamente mejor que el anterior se le convierte en el punto actual, en caso contrario se elige algún otro vecino. Según cómo se implante la rutina `SelectSteepestArea()` se obtienen las distintas variedades de los métodos de escalada. El procedimiento termina si no es posible realizar una nueva mejora.

Todas estas técnicas analíticas tienen dos serias desventajas, que van en detrimento de su robustez. Primeramente, todas admiten de un modo u otro que aunque no se pueda dar una forma cerrada a la función objetivo, tiene sentido el concepto de 'pendiente'; ello restringe grandemente su campo de aplicación. A decir verdad, se han desarrollado técnicas analíticas de búsqueda que no hacen uso de dicho concepto (la más conocida es la búsqueda poliédrica de Nelder-Meade, ver [BAZA93]), pero aun así siguen requiriendo que en todo momento esté completamente definida una *dirección preferente de búsqueda*. En definitiva, por una causa o por otra se precisa una gran cantidad de conocimiento específico. En segundo lugar, las técnicas analíticas son esencialmente locales, esto es, sólo pueden hallar subóptimos y, lo que es peor, en el caso de que haya varios (a esto se le llama *multimodalidad*) el acabar en uno u otro depende del punto inicial, siendo imposible determinar a priori a qué subóptimo llevará un punto inicial dado.

Esos dos resultados se resumen en que, por un lado, las técnicas analíticas de optimización son muy eficientes pero muy específicas, esto es, muy fuertes. Por otro lado, las desventajas de la alta especificidad no se logran compensar con la alta eficiencia, lo que las hace poco robustas.

```
BeginAlgorithm{IteratedHillclimber}
 Xcur = SelectStartingPoint();            Habitualmente se toma al
                                          azar
 for( t = 0; t < MaxIter; t++ )
  DONE = FALSE;
  Fcur = Eval(Xcur);
  P[t] = SelectSteepestArea(Xcur,n);
  while( not DONE )                       Se busca en la vecindad del
                                          punto actual
   Xnew = ChooseNeighbour(P[t]);
   Fnew = Eval(Xnew);
   if( Fnew > Fcur )
    Xcur = Xnew;
    Fcur = Fnew;
   else
    DONE = TRUE;
   endif
  endwhile
  if( ChkTermCond(Xcur, Fcur) )
   return;
 }
EndAlgorithm
```

Figura 1.3: Estructura general de un algoritmo de escalada

Técnicas enumerativas de búsqueda y optimización

El campo natural de aplicación de las *técnicas enumerativas* de optimización es el de los problemas combinatorios, aunque no de modo exclusivo. El método del símplex, sin ir más lejos, es una técnica enumerativa para resolver problemas de optimización numérica y la Programación Dinámica de Bellman sirve tanto para problemas numéricos como combinatorios.

No existen métodos generales para resolver problemas de optimización combinatoria, ni siquiera existe un problema típico. Debido en gran parte a esto, todas las técnicas enumerativas de búsqueda y optimización requieren gran cantidad de conocimiento específico.

Las técnicas de búsqueda en grafos constituyen las herramientas más comunes para la búsqueda enumerativa. Procedimientos como el algoritmo $A^\star$ o el de *ramificación&acotación* dan resultados bastante aceptables al resolver problemas combinatorios de mediano tamaño. Sin embargo, la mayoría de las técnicas clásicas para resolver problemas combinatorios tienen complejidad exponencial —esto es, el número de operaciones a realizar aumenta exponencialmente con el tamaño del problema—, por lo que resultan ser inútiles en la resolución de problemas grandes.

En definitiva, bien sea por su tendencia al crecimiento exponencial o por su alta solicitación de conocimiento específico, las técnicas enumerativas de optimización resultan ser poco robustas.

La conclusión final que se extrae de este apartado es que las técnicas clásicas de optimización, bien por demasiado específicas bien por demasiado ineficientes, no tienen gran robustez. Precisamente, la computación evolutiva trata de incrementar la robustez de los métodos de búsqueda añadiéndoles eficacia manteniendo su generalidad y viceversa.

Para ilustrar la discusión anterior se ha resuelto un conocido problema de minimización de una función particularmente difícil utilizando un método clásico de escalada en la dirección del máximo gradiente y un método evolutivo muy sencillo —estrategia evolutiva simple—. La función a minimizar es $f(x, y) = 100(y - x^2)^2 + (1 - x)^2$ y la dificultad estriba en que adquiere la forma de una profunda hendidura curvada en torno al mínimo, como puede verse en la figura 1.4 (las curvas de nivel tienen escala logarítmica para mayor claridad). Debido a esa peculiar forma, todos los métodos de escalada tienen una acusada tendencia a oscilar en torno a la zona más profunda de la hendidura; eso no sólo ralentiza la búsqueda sino que la hace más sensible a los errores en las evaluaciones o a las condiciones iniciales. Por contra, la estrategia evolutiva, pese a requerir menos información sobre la función $f(x, y)$, encuentra con más seguridad la dirección correcta de búsqueda. De este modo queda patente la importancia de utilizar métodos robustos de búsqueda para resolver problemas reales.

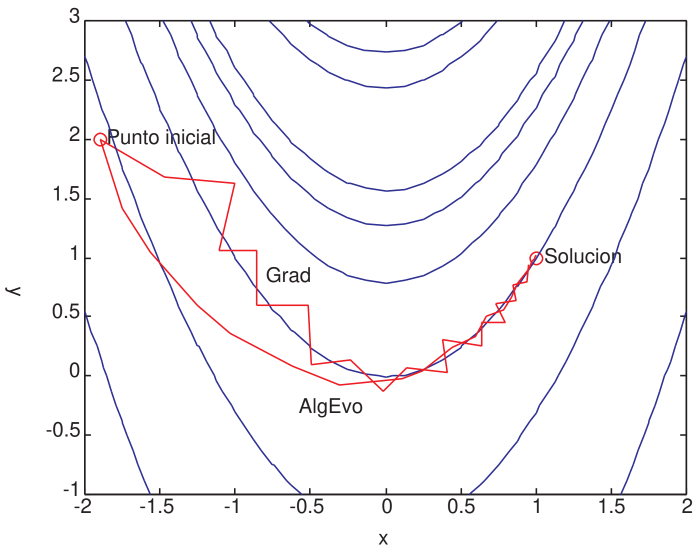

Figura 1.4: Función de Rosenbrock

Capítulo 2

Algoritmos Genéticos

De entre los cuatro paradigmas principales de la Computación Evolutiva, los Algoritmos Genéticos (en lo sucesivo, AGs) ocupan el lugar central. Las razones de esta preeminencia son tanto teóricas como prácticas, siendo las más importantes:

- ⋆ Los AGs constituyen el paradigma más completo de la Computación Evolutiva, esto es, reunen de modo natural, todas las ideas fundamentales de dicho enfoque.

- ⋆ Son muy flexibles, es decir, pueden adoptar con facilidad nuevas ideas, generales o específicas, que surjan en el campo de la CE. Además, se pueden hibridar fácilmente con otros paradigmas y enfoques, aunque no tengan ninguna relación con la CE.

- ⋆ Los AGs son el paradigma con mayor base teórica de entre los de la CE. Además, dicha base teórica es sencilla en su desarrollo y con grandes posibilidades de ampliación.

- ⋆ De entre todos los paradigmas de la CE son los que menos conocimiento específico necesitan para su funcionamiento, y en consecuencia, los más versátiles. Pero es que además pueden incorporar conocimiento específico con poco esfuerzo adicional.

- ⋆ Son fácilmente implantables en computadores de capacidades medias, proporcionando resultados francamente aceptables, en cuanto a precisión y recursos empleados, para una gran cantidad de problemas difícilmente resolubles por otros métodos.

- ⋆ Posiblemente como consecuencia de los anteriores argumentos los AGs son, con diferencia, el paradigma más usado de entre los de la CE y, junto con las Redes Neuronales, los más usados de toda la Computación Natural. Esto es importante en la medida en que pone a disposición del usuario gran cantidad de ensayos empíricos previos que proporcionan operadores, parámetros e implantaciones específicas para una amplia gama de problemas.

Debido a todo esto, los Algoritmos Evolutivos son, por defecto, algoritmos genéticos, y todo progreso en el campo de la CE se plantea teniendo en mente este paradigma. La consecuencia final es que es imprescindible realizar un estudio amplio y pormenorizado de tales métodos si se quiere tener un conocimiento cabal de la CE.

Una definición general de los AGs podría ser como sigue:

> Los Algoritmos Genéticos son métodos estocásticos de búsqueda ciega de soluciones cuasi-óptimas. En ellos se mantiene una población que representa a un conjunto de posibles soluciones la cual es sometida a ciertas transformaciones con las que se trata de obtener nuevos candidatos y a un proceso de selección sesgado en favor de los mejores candidatos.

Sin entrar de momento en pormenores, se debe señalar que la principal innovación de los AGs en el dominio de los métodos de búsqueda es la adición de un mecanismo de *selección* de soluciones. Recuérdese que la selección tiene dos vertientes: a corto plazo los mejores tienen más posibilidades de sobrevivir y a largo plazo los mejores tienen más posibilidades de tener descendencia. A causa de esto, el mecanismo de selección se desdobla en dos: uno elige los elementos que se van a trasformar posteriormente, este es el operador de *selección* a secas; el otro operador elige los elementos que van a sobrevivir, por lo que se le llama *reemplazo*.

En cuanto a las restantes características, cabe destacar que los AGs son métodos de búsqueda:

- **ciega**, es decir, no disponen de ningún conocimiento específico del problema, de manera que la búsqueda se basa exclusivamente en los valores de la función objetivo.
- **codificada**, puesto que no trabajan directamente sobre el dominio del problema, sino sobre representaciones de sus elementos.
- **múltiple**, esto es, buscan simultáneamente entre un conjunto de candidatos.
- **estocástica** referida tanto a las fases de selección como a las de transformación. Ello proporciona control sobre el factor de penetración de la búsqueda.

Todas las características enumeradas se introducen deliberadamente para proporcionar más robustez a la búsqueda, esto es, para darle más eficiencia sin perder la generalidad y viceversa. Goldberg [GOLD89,p22] justifica esta afirmación del siguiente modo:

> Los algoritmos genéticos manejan variables de decisión o de control representadas como cadenas con el fin de explotar similitudes entre

cadenas de altas prestaciones. Otros métodos tratan habitualmente con las funciones y sus variables de control directamente. Dado que los algoritmos genéticos operan en el nivel de códigos, son difíciles de engañar aun cuando la función sea difícil para los enfoques tradicionales.

Los algoritmos genéticos trabajan con una población; muchos otros métodos trabajan con un único punto. De este modo, los AGs encuentran seguridad en la cantidad. Al mantener una población de puntos bien adaptados, se reduce la probabilidad de alcanzar un falso óptimo.

Los algoritmos genéticos consiguen gran parte de su amplitud ignorando la información que no sea la del objetivo. Otros métodos se basan fuertemente en tal información, y en problemas donde la información necesaria no está disponible o es difícil de conseguir, estos otros métodos fallan. Los AGs son generales porque explotan la información disponible en cualquier problema de búsqueda. Los algoritmos genéticos procesan similitudes en el código subyacente junto con información proviniente de la ordenación de las estructuras de acuerdo con sus capacidades de supervivencia en el entorno actual. Al explotar una información tan fácilmente disponible, los AGs se pueden aplicar en prácticamente cualquier problema.

Las reglas de transición de los algoritmos genéticos son estocásticas; otros muchos métodos tienen reglas de transición deterministas. Hay una diferencia, no obstante, entre los operadores estocásticos de los algoritmos genéticos y otros métodos que no son más que paseos aleatorios. Los algoritmos genéticos usan el azar para guiar una búsqueda fuertemente explotadora. Esto puede parecer inusual, usar del azar para conseguir resultados concretos (los mejores puntos), pero hay gran cantidad de precedentes en la naturaleza.

Obsérvese que no sólo se ha justificado la introducción de las diversas características de los AGs en virtud de su robustez, sino que además se les compara en dichos términos con las técnicas clásicas del capítulo anterior.

Sin embargo, la característica esencial de los AGs no se observa directamente: es su capacidad de intercambio estructurado de información en paralelo, o abreviadamente el *paralelismo implícito*. Su significado concreto se explicará con mayor detalle en próximos apartados. De modo intuitivo, esta característica se refiere a lo siguiente: los AGs procesan externamente cadenas de códigos, sin embargo, lo que se está procesando internamente son similitudes entre cadenas y de manera tal que al procesar cada una de las cadenas de la población se están procesando a la vez todos los patrones de similitud que contienen, que son muchos más. Es precisamente por esta propiedad que los AGs son mucho más eficaces que otros métodos de búsqueda ciega, y por tanto más robustos con independencia de otras consideraciones.

2.1 Estructura y componentes básicos

Como ya quedó dicho en el capítulo anterior, al ajecutar un Algoritmo Evolutivo una población de individuos, que representan a un conjunto de candidatos a soluciones de un problema, es sometida a una serie de transformaciones con las que se actualiza la búsqueda y después a un proceso de selección que favorece a los mejores individuos. Cada ciclo de selección+búsqueda constituye una generación, y se espera del AE que, al cabo de un número razonable de generaciones, el mejor individuo represente a un candidato lo suficientemente próximo a la solución buscada. Los Algoritmos Genéticos, como ejemplo fundamental de algoritmos evolutivos, siguen dicho esquema básico con ciertas peculiaridades propias de la implantación que se verán seguidamente.

De modo gráfico, la estructura genérica del bucle básico de un AG se sintetiza en el siguiente diagrama:

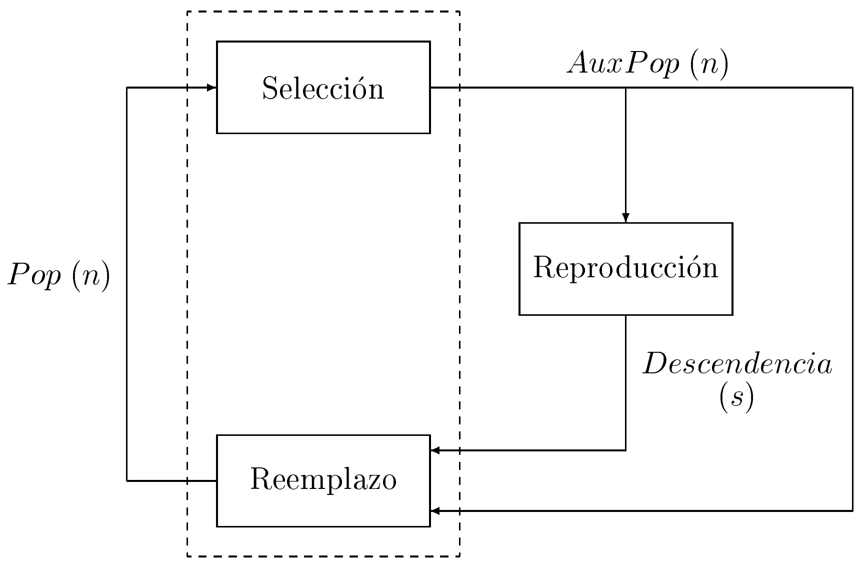

Figura 2.1: Bucle básico de un AG

Como puede verse, una población *Pop*, que consta de n miembros se somete a un proceso de *selección* para constituir una población intermedia *AuxPop* de n *criadores*. De dicha población intermedia se extrae un grupo reducido de individuos llamados *progenitores* que son los que efectivamente se van a reproducir. Sirviéndose de los *operadores genéticos*, los progenitores son sometidos a ciertas transformaciones de *alteración* y *recombinación* en la fase de *reproducción* en virtud de las cuales se generan s nuevos individuos que constituyen la *Descendencia*. Para formar la nueva población $Pop[t+1]$ se deben seleccionar n supervivientes de entre los $n + s$ de la población auxiliar y la descendencia, eso se hace en la fase de *reemplazo*. Como ya se comentó, la selección (recuadro rayado) se hace en dos etapas con la idea de emular las dos vertientes del Principio de Selección Natural: selección de criadores o 'selección' a secas y selección de supervivientes para la próxima generación o 'reemplazo'.

Los objetos que se someten a evolución en un AG son cadenas binarias $\mathbf{v}$ sobre las que se codifican los elementos $\mathbf{x}$ del espacio de búsqueda; ambos reciben el nombre de *individuo*, *codificado* el uno y *sin codificar* el otro. También son usuales las denominaciones 'ejemplar', 'muestra' o 'punto'.

Cada individuo $\mathbf{x}$ consta de *m posiciones* (en la bibliografía inglesa *loc-us/i*) que son ocupadas por otros tantos atributos o *genes* los cuales se codifican en $\ell = L_1 + \ldots + L_m$ bits. Naturalmente, si un atributo puede tomar más de dos valores o *alelos* se deberá representar mediante varios bits; de hecho es inmediato comprobar que si el j-ésimo gen consta de a_j alelos, entonces se deberá codificar mediante $L_j = \lceil \log_2 a_j \rceil$ bits. En este punto conviene distinguir entre la estructura de los individuos y el contenido de cada uno: la estructura se deriva de la *representación*, es igual para todos y se expresa así "los 4 (vg.) primeros bits representan al primer gen, los 8 siguientes al segundo, etc…", el contenido se deriva de la *codificación* y es simplemente el entero binario con que se codifica cierto individuo.
En la jerga al uso a la estructura de un individuo se le dice *genotipo* y a su contenido *fenotipo*. A la hora de programar un AG, el genotipo se implanta a través de una clase y los fenotipos a través de objetos o "instanciaciones" de dicha clase.

Todo el procedimiento de búsqueda está guiado exclusivamente por una *función de aptitud* $u(\mathbf{x})$, la cual se obtiene directamente a partir de la *función de evaluación* $f(\mathbf{x})$ del correspondiente problema. La función de evaluación no tiene por qué estar expresada de forma cerrada como una función objetivo clásica; basta con que proporcione un 'índice de idoneidad' para cada uno de los candidatos a solución que se le presenten. De todos modos, es conveniente que el procedimiento de obtención de dicho índice se pueda implantar con facilidad en un computador, dado que la evolución de todo el algoritmo va a depender de él. Comúnmente, a los valores proporcionados por la función de evaluación se les dice *evaluaciones* o bien *aptitudes brutas* mientras que a los valores proporcionados por la función de aptitud se les dice *aptitudes* a secas, y también, para distinguirlas de las aptitudes brutas, *aptitudes netas*.

En la tabla 2.1 se muestra la notación que se va a usar en el desarrollo del presente capítulo. Para mayor comodidad es conveniente utilizar una notación doble: abreviada y desplegada. Con el fin de evitar confusiones se ha decidido utilizar en la medida de lo posible la notación abreviada para los individuos sin codificar, $\mathbf{x}$, y la notación desplegada para los individuos codificados, $\mathbf{v}$.

Notación		Significado
Abreviada	Desplegada	
$\mathbf{x}$	$\mathbf{v}$	Individuo
$f(\mathbf{x})$	$Eval(\mathbf{v})$	Función de evaluación
$u(\mathbf{x})$	$Fitness(\mathbf{v})$	Función de aptitud
$p(\mathbf{x})$	$Score(\mathbf{v})$	Puntuación
$q(\mathbf{x})$	$AccScore(\mathbf{v})$	Puntuación acumulada
n	$PopSize$	Tamaño de la población
ℓ	Len	Tamaño del individuo
m	Dim	Número de genes
L_j	L_j	Tamaño del j-ésimo gen
r	$EliteSize$	Tamaño de la élite
s	$OffsprSize$	Tamaño de la descendencia

Tabla 2.1: Notación

2.1.1 Métodos y criterios necesarios para implantar un AG

Para implantar un AG es necesario definir de modo inequívoco los siguientes métodos y criterios.

1. ***Criterio de codificación:*** Dado que los AGs operan exclusivamente con cadenas binarias (*representación*) se debe especificar el procedimiento (*codificación*) con el que se hace corresponder cada punto del dominio del problema con una cadena, o con la terminología anterior, el mecanismo de paso del genotipo a los fenotipos.

2. ***Criterio de tratamiento de los individuos no factibles:*** No siempre es posible establecer una correspondencia punto por punto entre el dominio de un problema y el conjunto de las cadenas binarias de tamaño ℓ usadas para resolverlo, ie., el espacio de búsqueda. Como consecuencia, no todas las cadenas codifican elemento válidos del espacio de búsqueda y se deben habilitar procedimientos útiles para distinguirlas.

3. ***Criterio de inicialización:*** Se refiere a cómo se debe construir la población inicial con la que se arranca el bucle básico del AG.

4. ***Criterio de parada:*** Se deben concretar las condiciones con las que se considera que el AG ha dado con una solución aceptable o, en su defecto, ha fracasado en la búsqueda y no tiene sentido continuar.

5. ***Funciones de evaluación y aptitud:*** Se debe concretar la función de evaluación más apropiada para el problema, así como la función de aptitud que utilizará el AG para resolverlo.

6. ***Operadores genéticos:*** Se llaman así a los operadores con los que se lleva a cabo la reproducción. Todo AG hace uso de al menos dos operadores genéticos, el cruce y la mutación; no obstante ellos no son los únicos posibles y además admiten variaciones.

7. ***Criterios de selección:*** La selección debe dirigir el proceso de búsqueda en favor de los miembros más aptos, pero existen muchas maneras de llevar esto a cabo.

8. ***Criterios de reemplazo:*** Los criterios con que se seleccionan los criadores no necesariamente han de ser los mismos con que se seleccionan los supervivientes, de ahí la necesidad de especificarlos por separado.

9. ***Parámetros de funcionamiento:*** Un AG necesita que se le proporcionen ciertos parámetros de funcionamiento, tales como el tamaño de la población, las probabilidades de aplicación de los operadores genéticos, la precisión de la codificación, las tolerancias de la convergencia, etc...

En el capítulo 3 se describirán razonadamente todas las posibilidades de elección de esos nueve elementos. La elección de unas u otras no sólo depende del tipo de problema a resolver, sino también de que no se violen ciertos requisitos imprescindibles para el buen funcionamiento del AG.

2.1.2 Mecanismos de muestreo de poblaciones

Un mecanismo auxiliar pero fundamental para los AGs es el de *muestreo de poblaciones*, esto es, la elección según ciertos criterios de un subconjunto de k individuos de una población especificada. Los mecanismos de muestreo son muy variados, distinguiéndose tres grupos fundamentales según el grado de intervención del azar en el proceso:

1. ***Muestreo directo:*** Se toma un subconjunto de individuos de la población siguiendo un criterio fijo, del estilo de “los k mejores”, “los k peores”, “a dedo”, etc...

2. ***Muestreo aleatorio simple o equiprobable:*** Se asignan a todos los elementos de la población base las mismas probabilidades de formar parte de la muestra y se constituye ésta mediante ensayos de Bernuolli simples.

3. ***Muestreos estocásticos:*** Se asignan probabilidades de selección o *puntuaciones* a los elementos de la población base en función (directa o indirecta) de su aptitud. Por defecto, la puntuación p_i asociada al individuo

$\mathbf{x}_i$ de la población $\mathcal{P} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ se calcula como la aptitud relativa de dicho individuo; esto es, siendo $u_1, \ldots, u_n$ las respectivas aptitudes se tiene que

$$p_i \stackrel{\text{def}}{=} \frac{u_i}{u_1 + \ldots + u_n} \qquad (\forall i = 1, \ldots, n)$$

así se garantiza que $p_1 + \ldots + p_n = 1$, tal como corresponde a una distribución de probabilidades.
La muestra se constituye de modo estocástico haciendo uso de dichas puntuaciones.

Existen muchos mecanismos de muestreo estocástico según para lo que se apliquen. En concreto, al implantar algoritmos genéticos se usan fundamentalmente cuatro tipos de muestreo estocástico, que se describirán seguidamente. Para ilustrarlos, se considerará el problema de muestrear estocásticamente dos elementos de la población $\mathcal{P} = \{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4\}$ siendo las puntuaciones asociadas $p_1 = 0.1$, $p_2 = 0.3$, $p_3 = 0.1$ y $p_4 = 0.5$ respectivamente.

- ***Por sorteo:*** Se consideran las puntuaciones estrictamente como probabilidades de elección para formar la muestra, y se constituye ésta realizando k ensayos de una variable aleatoria con dicha distribución de probabilidades. Dado que habitualmente sólo se dispone de un generador de números aleatorios simples (ie., uniformemente distribuidos entre 0 y 1), para simular un ensayo de la distribución $\{p_1, \ldots, p_n\}$ se hace lo siguiente:

  1. Se calculan las *puntuaciones acumuladas* así:

  $$\begin{aligned} q_0 &:= 0 \\ q_i &:= p_1 + \ldots + p_i \qquad (\forall i = 1, \ldots, n) \end{aligned}$$

  2. Se genera un número aleatorio simple $r \leftarrow \text{URand}[0, 1)$
  3. Se elige el individuo $\mathbf{x}_i$ que verifique

  $$q_{i-1} < r < q_i$$

  Para muestrear por sorteo k individuos, se modifican trivialmente los dos últimos pasos así:

  2. Se generan k números aleatorios simples $r_j \leftarrow \text{URand}[0, 1) \quad (\forall j = 1, \ldots, k)$.
  3. Para cada $j = 1, \ldots, k$ se elige el individuo $\mathbf{x}_i$ que verifique

  $$q_{i-1} < r_j < q_i$$

Para el ejemplo propuesto las puntuaciones acumuladas son

$$q_1 = 0.1 \quad q_2 = 0.4 \qquad q_3 = 0.5 \text{ y } q_4 = 1.0$$

respectivamente, y los $k = 2$ números aleatorios simples se han hallado como

$$r_1 = 0.02433 \quad \text{y} \quad r_2 = 0.51772$$

De esta manera, el primer elemento de la muestra resulta ser $\mathbf{x}_1$ dado que $q_0 < r_1 < q_1$ y el segundo será $\mathbf{x}_4$ puesto que $q_3 < r_1 < q_4$.
Nótese que existe la posibilidad de que algunos individuos puedan ser elegidos varias veces en una muestra (los que tengan mayor puntuación o un golpe de buena suerte con los r_j) y que otros individuos pueden no ser elegidos nunca (los que tengan menor puntuación o mala suerte).

Debido a sus buenas propiedades teóricas, este es el muestreo que se utiliza por defecto con los AGs. No obstante, por diversos motivos que se verán en 3.2.7, sus prestaciones en la práctica no están siempre a la altura de las expectativas teóricas.

- ***Por restos:*** A cada individuo $\mathbf{x}_i$ se le asignan directamente $\lfloor p_i \cdot k \rfloor$ puestos en la muestra. Seguidamente los individuos se reparten los puestos vacantes en función de sus puntuaciones. El reparto se suele hacer por sorteo (ver anterior apartado) y entonces se le dice "muestreo estocástico por restos".
Para el ejemplo propuesto, a $\mathbf{x}_4$ le corresponde un puesto en la muestra; el individuo que ocupe el otro puesto se elige por sorteo, ie, se genera un número aleatorio simple $r = 0.39874$ y se compara con las puntuaciones acumuladas q_i, en este caso el puesto vacante le corresponde a $\mathbf{x}_2$ dado que $q_1 < r < q_2$.
Existe una variante en la que el reparto de las vacantes se hace por muestreo directo; nótese que en esta variante no hay ninguna intervención del azar, ie., no es un método de muestreo estocástico sino directo, por este motivo se le llama "muestreo determinista por restos", para distinguirle del descrito anteriormente. Obsérvese que este último método de muestreo no es otra cosa que la conocida regla de repartos enteros de d'Hont.

- ***Universal o por ruleta:*** Es análogo al muestreo por sorteo sólo que ahora se genera un único número aleatorio simple r y con él se asignan todas las muestras de modo parecido a como se haría al girar una ruleta (ver figura 2.2).

Para simular una tirada de ruleta se definen a partir de $r \leftarrow \text{URand}[0, 1)$ los siguientes números:

$$r_j := \frac{r + j - 1}{k} \qquad (\forall j = 1, \ldots, k)$$

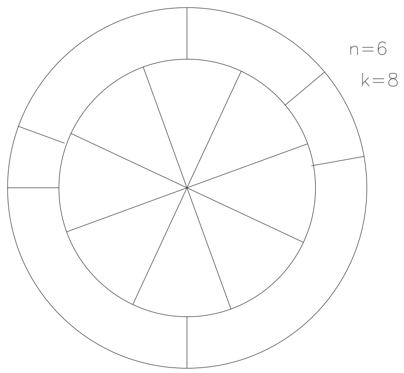

Figura 2.2: Muestreo por ruleta

Con eso ya se determina la muestra comparando los r_j con las puntuaciones acumuladas a la manera habitual.
Para el ejemplo propuesto, si el número aleatorio es $r = 0.39874$ entonces se tiene:

$$r_1 = 0.19937 \qquad r_2 = 0.69937$$

lo que proporciona la muestra $\{\mathbf{x}_2, \mathbf{x}_4\}$.

Como puede verse, la implantación del muestreo estocástico universal es muy sencilla y rápida, y en la práctica proporciona unas prestaciones análogas a las del muestreo por sorteo; por este motivo se suele usar en sustitución de éste.
A propósito de todo ésto, en la bibliografía antigua se suele llamar muestreo por ruleta (*roulette wheel sampling*) a lo que aquí se ha llamado muestreo por sorteo (*lottery sampling*), y muestreo por sorteo a lo que aquí se ha llamado muestreo universal. En la presente memoria no se usará esa terminología.

- ***Por torneos:*** Cada elemento de la muestra se toma eligiendo el mejor de los individuos de un conjunto de z elementos tomados al azar de la población base; esto se repite k veces hasta completar la muestra. El parámetro z suele ser un entero pequeño comparado con el tamaño de la población base, normalmente 2 ó 3. Nótese que en este caso no es necesario hacer la asignación de puntuaciones.

Todos los mecanismos de muestreo vistos admiten algunas variantes no necesariamente excluyentes; las más empleadas al trabajar con algoritmos genéticos son estas tres:

1. ***muestreo diferenciado:*** Cada elemento de la población base se puede tomar para formar la muestra a lo sumo una vez.

2. ***muestreo conservador:*** Todos los elementos de la población base tienen alguna oportunidad —ie., probabilidad no nula— de ser elegidos. También se le conoce como 'muestreo duro'.

3. ***muestreo excluyente:*** Se excluyen a priori algunos individuos del proceso de muestreo. También se le llama 'muestreo extintivo'.

De esta manera se habla, por ejemplo, de que el proceso de selección de criadores de cierto AG se ha implantado a través de un 'muestreo estocástico por torneos de tamaño 2 en la variedad conservadora'.

2.1.3 Procedimientos básicos de un AG

Una vez vistos los principales mecanismos de muestreo, se pueden describir los tres procedimientos básicos de que consta un AG, a saber: selección (de criadores), reproducción (o transformación) y reemplazo (o selección de supervivientes). Nótese que para implantar un AG se requiere un mínimo de dos parámetros: el tamaño de la población n y el tamaño de la descendencia s.

El proceso de selección: Consiste en muestrear, a partir de la población inicial, los n elementos de la población de criadores. El criterio concreto de muestreo depende del problema y del buen juicio del programador. Los más usados en la práctica son los muestreos por sorteo, universal y por torneos, en sus variedades conservadoras. Los muestreos deterministas se usan muy poco, entre otros motivos porque van contra la filosofía del método.

El proceso de reproducción: Aplicando los operadores de transformación (cruce, mutación, inversión, ...) sobre ciertos miembros de la población de criadores se obtiene una descendencia de s nuevos miembros. La determinación exacta de qué miembros se van a reproducir, ie., de los progenitores, se suele realizar mediante ensayos de Bernuolli; se admite a veces la variedad excluyente. No es necesario —de hecho, no es habitual— dar por adelantado el valor de s, normalmente sólo se da su valor esperado de modo indirecto a través de las probabilidades de aplicación de los operadores genéticos. Cuanto más grande sea el valor (real o esperado) de s más variará la población de una generación a otra. Salvo indicación en contra, lo común es trabajar con valores de s no mayores del 60% de n.

Se verán a su debido tiempo los diversos operadores genéticos que se usan con los AGs. Ahora bien, existen dos grupos de operadores que nunca faltan en un algoritmo genético: el cruce y la mutación.

Los operadores de cruce son el arquetipo de operadores de *recombinación*: actúan sobre parejas de individuos y normalmente originan otro par de individuos que combinan características de los progenitores. Dado que en los AGs los individuos están representados a través de cadenas binarias, el cruce se lleva a cabo por intercambio de segmentos, a la manera indicada en la figura.

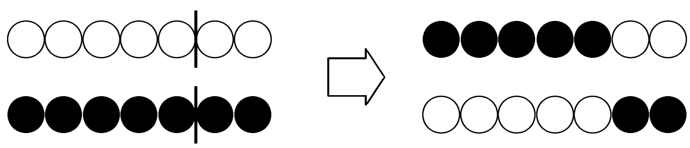

Figura 2.3: Cruce

Los operadores de mutación, por su parte, son el arquetipo de operadores de *alteración* dado que actúan sobre individuos solos, realizando una pequeña modificación alguno de sus genes o en el conjunto.

El motivo de hacer esta separación es el siguiente: entendiendo que la búsqueda propiamente dicha se lleva a cabo en la fase de reproducción, resulta conveniente diferenciar los operadores de búsqueda en profundidad de los de búsqueda en anchura; los primeros se encargarán de explotar las mejores características de que disponga la población actual, los otros se encargarán de explorar nuevos dominios en busca de mejores soluciones. Desde esta perspectiva, los operadores de cruce son los que principalmente se encargan de la búsqueda en profundidad y los de mutación de la búsqueda en anchura. Así, dando mayor importancia a unos o a otros se puede ajustar el tipo de búsqueda a las necesidades del problema.
Sin necesidad de entrar de momento en mayores precisiones es evidente que la división tiene un valor práctico para diferenciar dos modos complementarios de implantar la reproducción: el cruce se encarga de realizar un intercambio estructurado de información, la mutación proporciona una garantía de accesibilidad para todos los puntos del espacio de búsqueda. Se volverá a comentar este tema en 2.3.3 y 3.2.6.

Existe una variante del proceso de reproducción que recibe el nombre de ***reproducción operacional***; en ella cada miembro de la descendencia es producido por un solo operador genético y además éstos operadores sólo pueden actuar sobre los individuos, no sobre las poblaciones o sobre los genes.

El proceso de reemplazo: A partir de los n miembros de la población de criadores y de los s miembros de la población de descendientes se debe obtener una nueva población de n miembros. Para hacerlo existen varios criterios:

1. ***Reemplazo inmediato (o al vuelo):*** Los s descendientes sustituyen a sus respectivos progenitores.

2. ***Reemplazo con factor de llenado:*** Los s descendientes sustituyen a aquéllos miembros de la población de criadores que más se les parezcan.

3. ***Reemplazo por inserción (o de tipo 'coma'):*** Según el tamaño relativo de la descendencia respecto de la población se distinguen dos casos,

   1. $s \leq n$: Se muestrean para ser eliminados s miembros de la población de criadores (según cierto criterio; normalmente, los peores). Esos miembros serán sustituidos por los descendientes.
   2. $s > n$: Se muestrean n miembros de la población de descendientes y se constituye con ellos la nueva población. Nótese que de este modo cualquier individuo sólo puede vivir a lo sumo una generación.

4. ***Reemplazo por inclusión (o de tipo 'más'):*** Se juntan los s descendientes con los n progenitores en una sóla población, y en ella se muestrean n miembros (normalmente, los mejores).

Naturalmente, los dos primeros criterios sólo se pueden usar cuando $s \leq n$.

Sin hacer de esto una cuestión de principios, una buena elección de criterios para un algoritmo genético incorpora la selección mediante muestreo estocástico universal en la variedad conservadora; el reemplazo suele ser inmediato aunque es preferible el reemplazo por inserción con el mismo tipo de muestreo.

Considerando todo lo discutido en esta sección, se muestra a continuación un seudocódigo con el que implantar un algoritmo genético:

```
BeginAlgorithm{GeneticAlgorithm}
 P[0] = InitPop();
 FitP[0] = EvalPop(P[0]);
 for( t = 0; t < MaxNumGen; t++){
  Q[t] = SelectBreedersFrom(P[t]);        Población de criadores
  Q[t] = Crossover(Q[t]);
  Q[t] = Mutate(Q[t]);
  FitQ[t] = EvalPop(Q[t]);
  P[t] = SelSurviv(P[t], Q[t], FitP[t], FitQ[t]);
  FitP[t] = EvalPop(P[t]);
  if( ChkTermCond(P[t], FitP[t]) )
   return;
 }
EndAlgorithm
```

2.2 El AG básico (SGA). Implantación y ejemplos

En esta sección se estudiará una implantación elemental pero completa del algoritmo descrito anteriormente y se resolverá con ella un ejemplo práctico de optimización paramétrica multimodal. Se trata, por un lado, de aclarar y concretar todo lo discutido en la sección anterior referido a los métodos, y por el otro lado facilitar la comprensión del estudio teórico de la siguiente sección.

Dicha implantación fue propuesta por Goldberg en su texto clásico [GOLD89] y es la que, con cambios menores, se usa habitualmente para ilustrar el funcionamiento de los AGs.

El SGA (*simple genetic algorithm*) es una algoritmo genético que incorpora los siguientes métodos y criterios:

1. *Criterio de codificación:* Específico de cada problema. Debe hacer corresponder a cada punto del dominio del problema un elemento del espacio de búsqueda, el cual necesariamente consta de enteros binarios sin signo, esto es, cadenas binarias simples.
2. *Criterio de tratamiento de los individuos no factibles:* No hay. Se considera que la codificación se hace de tal manera que todas las cadenas posibles representan a individuos factibles.
3. *Criterio de inicialización:* La población inicial consta de cadenas binarias elegidas al azar.
4. *Funciones de evaluación y aptitud:* La función de aptitud coincide con la de evaluación. Preferiblemente la función de evaluación viene dada a través de una función objetivo.
5. *Operadores genéticos:* Cruce monopunto y mutación bit a bit sobre los individuos codificados.
6. *Criterio de selección:* Por sorteo.
7. *Criterio de reemplazo:* Inmediato.
8. *Criterio de parada:* Fijando el número máximo de iteraciones.
9. *Parámetros de funcionamiento:* Discrecionales. Una elección típica [GREF90] es

$$PopSize = 30 \quad MaxIter = 50 \quad p_{xov} = 60\% \quad p_{mut} = 3.33\%$$

De todas esas especificaciones las únicas que precisan explicación son las referidas a los operadores genéticos. El *cruce monopunto* genera dos descendientes a partir de dos progenitores cortándolos en una posición elegida al azar e intercambiando los respectivos segmentos. Es decir, dados dos progenitores codificados como $\mathbf{v} = \langle b_1 \dots b_\ell \rangle$ y $\mathbf{w} = \langle c_1 \dots c_\ell \rangle$ con $b_j, c_j \in 0, 1$ $(\forall j = 1, \dots, \ell)$ se genera un número aleatorio $r \leftarrow \mathrm{URandInt}[1, \ell - 1]$ y se procede así:

$$\begin{array}{ccc} \langle b_1, \ldots, b_{pos-1} | b_{pos}, \ldots, b_\ell \rangle & & \langle b_1, \ldots, b_{pos-1} | c_{pos}, \ldots, c_\ell \rangle \\ & \Longrightarrow & \\ \langle c_1, \ldots, c_{pos-1} | c_{pos}, \ldots, c_\ell \rangle & & \langle c_1, \ldots, c_{pos-1} | b_{pos}, \ldots, b_\ell \rangle \end{array}$$

Cada aplicación de la *mutación bit a bit* conmuta un bit de entre todos los $(n \times \ell)$ de la población.

Como ya se comentó en 2.1.3, la determinación exacta de qué individuos se van a cruzar y qué bits se van mutar se lleva a cabo mediante ensayos de Bernuolli. Para el cruce el proceso es así:

1. Para cada individuo se genera un número aleatorio $r_i \leftarrow \mathrm{URand}[0, 1)$.
2. Se comparan los r_i con la probabilidad de cruce, p_{xov}, que es un parámetro del método.
3. Son seleccionados para cruzarse todos los individuos $\mathbf{v}_i$ para los que

$$r_i < p_{xov}$$

Para la mutación se realiza el mismo proceso con cada uno de los $n \times \ell$ bits de la población y usando la probabilidad de mutación p_{mut}.

El emparejamiento de los individuos a cruzar se hace al azar; habitualmente, las etapas anteriores al cruce han introducido el suficiente desorden en la población como para que un emparejamiento sucesivo se pueda considerar aleatorio. Todavía queda un pequeño problema que resolver para implantar completamente el cruce: ¿Qué hacer cuando el número de individuos a cruzar sea impar? En tal caso, el último individuo quedará desemparejado y lo que se hace es bien eliminarlo o bien emparejarlo con cualquier otro elegido al azar.

2.2.1 Ejemplo práctico de implantación de un AG

Se va a resolver el siguiente problema de optimización paramétrica propuesto por R.L. Riolo en [RIOL92].

Hallar

$$x^\circ = \arg\max \left\{ f(x) = x + |\sin(32\,\pi\, x)| \;:\; x \in [0, 1] \right\}$$

utilizando un algoritmo genético de parámetros

$$PopSize = 6 \quad p_{xov} = 30\% \quad p_{mut} = 1\%$$

con una precisión de media centésima (ie., $TOL = 5 \times 10^{-3}$).

El problema en sí tiene la dificultad de ser fuertemente multimodal, ie., en el intervalo $x \in [0,1]$ la función objetivo posee 32 óptimos locales equiespaciados (ver figura 2.4). Por tal motivo, los métodos analíticos de optimización (vg. el de Newton–Raphson) manifiestan una acusada tendencia a atascarse en alguno de esos 32 subóptimos, y sólo convergen al óptimo global ($x = 0.98447395 \rightarrow f(x) = 1.98442447$) si se les da una aproximación inicial muy buena. Es por esto que se debe recurrir a otro tipo de algoritmos de búsqueda.

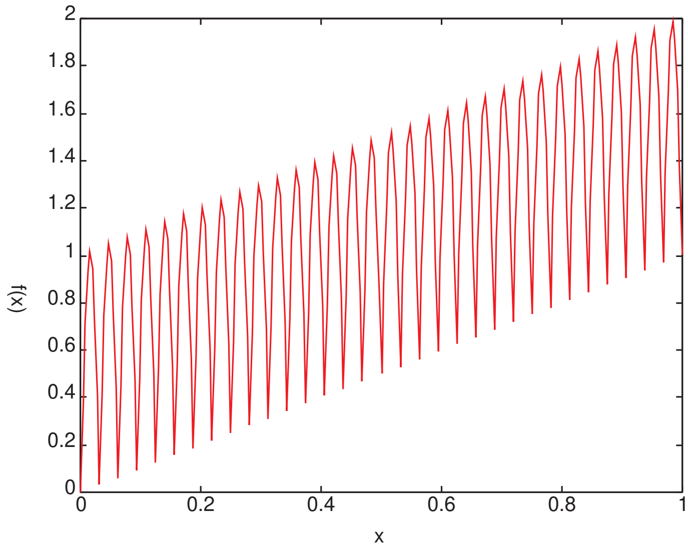

Figura 2.4: Función objetivo

Lo primero que hay que hacer es la *codificación*, esto es, se ha de representar convenientemente el dominio del problema a través de enteros binarios sin signo. La elección más sencilla consiste en discretizar el intervalo $[x_{min}, x_{max}]$ en una cantidad de puntos 2^{Len} tal que la distancia entre puntos consecutivos sea menor que la tolerancia especificada, ie.,

$$\frac{x_{max} - x_{min}}{2^{Len} - 1} < TOL$$

así cada punto del espacio de búsqueda queda representado mediante un entero binario de longitud Len, comenzando por `0...0` que representa a x_{min} y terminando en `1...1` que representa a x_{max}.

En definitiva, la clave de la representación reside en calcular la longitud de los individuos. Si se dispone directamente de la tolerancia TOL el cálculo es inmediato a partir de la fórmula anterior:

$$Len = \left\lceil \log_2 \left(1 + \frac{x_{max} - x_{min}}{TOL} \right) \right\rceil$$

Recíprocamente, se puede calcular el punto x que corresponde a un individuo $\mathbf{v}$ mediante la siguiente *fórmula de decodificación*:

$$x(\mathbf{v}) = x_{min} + \text{bin2dec}(\mathbf{v}) \cdot \frac{x_{max} - x_{min}}{2^{Len} - 1}$$

Para el presente caso se tiene

$$Len = \left\lceil \log_2 \left(1 + \frac{1-0}{0.005} \right) \right\rceil = 8$$

esto es, el intervalo $x \in [0, 1]$ se puede representar con precisión menor o igual de media centésima usando cadenas binarias de 8 bits de longitud. De hecho, dado que con ocho bits de longitud se pueden representar $2^8 = 256$ puntos del intervalo de búsqueda, la precisión será exactamente $1/256 = 0.00390625 (< TOL)$.

Otro paso previo es el de inicialización. En la siguiente tabla se representa la población inicial de cadenas —elegidas al azar— junto con los valores significativos asociados a cada una:

i	$\mathbf{v}_i$	x_i	$Eval(\mathbf{v}_i)$	$Fitness(\mathbf{v}_i)$	$Score(\mathbf{v}_i)$	$AccScore(\mathbf{v}_i)$
1	`10111010`	0.72656250	1.68308	1.68308	0.25820	0.25820
2	`10010111`	0.58984375	1.44967	1.44967	0.22239	0.48059
3	`00110101`	0.20703125	0.56235	0.56235	0.08627	0.56686
4	`11001101`	0.80078125	1.78708	1.78708	0.27415	0.95070
5	`01000100`	0.26562500	0.71500	0.71500	0.10969	0.95070
6	`00011110`	0.11718750	0.32126	0.32126	0.04930	1.00000

Obsérvese que al principio el individuo más apto es $\mathbf{v}_4 =$ `11001101` y el menos apto $\mathbf{v}_6 =$ `00011110`.
No se ha hecho ni escalado ni desplazamiento de la aptitud (ver apdo. 3.2.5.2), por lo que la función de aptitud, $Fitness(\mathbf{v}_i)$, coincide con la de evaluación, $Eval(\mathbf{v}_i)$, y esta con el objetivo $f(\mathbf{x})$.

El proceso a seguir es este:

1. Construir la tabla de valores asociados a la población actual: Es la que se acaba de calcular. Obsérvese que al principio el individuo más apto es $\mathbf{v}_4 =$ `11001101` y el menos apto $\mathbf{v}_6 =$ `00011110`. La aptitud global de la población se obtiene sumando las de los individuos que la componen y vale $Fitness(P[0]) = 6.51854092$.

2. Realizar el sorteo de individuos para reproducirse: Para ello se generan seis números aleatorios URand$[0, 1)$ y se comparan con las participaciones acumuladas $AccScore(\mathbf{v}_i)$. Seguidamente se muestran los números aleatorios obtenidos y los correspondientes individuos seleccionados

$$\begin{array}{lcl} 0.52920 & \rightarrow & \mathbf{v}_3 \\ 0.04358 & \rightarrow & \mathbf{v}_1 \\ 0.29492 & \rightarrow & \mathbf{v}_2 \\ 0.04109 & \rightarrow & \mathbf{v}_1 \\ 0.28483 & \rightarrow & \mathbf{v}_2 \\ 0.78687 & \rightarrow & \mathbf{v}_4 \end{array}$$

de tal suerte que la población intermedia es:

i	$\mathbf{v}'_i$
1	00110101
2	10111010
3	10010111
4	10111010
5	10010111
6	11001101

3. De entre los individuos anteriores se eligen primero los que se van a cruzar. Como $p_{xov} = 30\%$, en promedio se cruzarán $6 \times 30\% = 1.8$ individuos. Para determinar la cantidad correspondiente en cada iteración se procede así:
Se generan seis números aleatorios $r_i = \text{URand}[0, 1)$, uno para cada individuo, y se eligen para el cruce aquellos para los que $r_i < p_{xov}$. Por ejemplo, si los números son

0.72232 0.31575 0.56156 0.15608 0.86910 0.03070

entonces se seleccionan los individuos $\mathbf{v}'_4$ y $\mathbf{v}'_6$.
Con esos dos individuos se realiza el cruce. Si hubiera salido un número impar se habría tenido que elegir otro o desechar uno de ellos al azar.

4. Se cruzan todos los individuos elegidos en un punto al azar (ie., URand-Int$[1, Len - 1]$). Como hay dos, es trivial hacer el emparejamiento, si hubiera cuatro, se emparejarían arbitriariamente, vg., cada uno con el siguiente.
En el presente caso se ha obtenido como punto de cruce el 6º bit, lo que hace que $\mathbf{v}_4$ y $\mathbf{v}_6$ se transformen así:

$$\begin{array}{ccc} \texttt{101110|10} & & \mathbf{v}''_4 = \texttt{101110|01} \\ & \rightarrow & \\ \texttt{110011|01} & & \mathbf{v}''_6 = \texttt{110011|10} \end{array}$$

5. Esos nuevos individuos sustituyen a sus progenitores inmediatamente, por lo que la población intermedia es ahora:

i	$\mathbf{v}''_i$
1	`00110101`
2	`10111010`
3	`10010111`
4	`10111001`
5	`10010111`
6	`11001110`

6. Se seleccionan ahora los bits (¡no los individuos!) que se van a mutar. Como la población contiene $8 \times 6 = 48$ bits y $p_{mut} = 1\%$ en promedio mutarán 0.48 bits, lo cual quiere decir que casi la mitad de las veces se aplicará dicho operador.
Como todos los bits tienen las mismas probabilides de ser mutados, para cada uno de ellos se genera un $r_j = \text{URand}[0,1)$ y si $r_j < p_{mut}$ se muta dicho bit.

Se han generado aparte los 48 números aleatorios r_j y se ha visto que son menores que 0.01 el 13º y el 31º. No es muy probable que haya que mutar tantos bits (en promedio debería mutar uno cada dos ciclos), pero entra dentro de lo posible.

7. Se mutan los bits correspondientes y se obtiene definitivamente la nueva población:

i	$\mathbf{v}'''_i$
1	`00110101`
2	`10110010`
3	`10010111`
4	`10111011`
5	`10010111`
6	`11001110`

Con esto se completa una iteración —ie., una generación—. Se deberán repetir estos siete pasos tantas veces como indique el parámetro MaxIter. Aquí sólo se hará otra vez, para comprobar cómo ha mejorado la aptitud total de la población:

1. Tabla de valores significativos asociados a cada individuo de la población recién hallada:

i	$\mathbf{v}_i$	x_i	$Eval(\mathbf{v}_i)$	$Fitness(\mathbf{v}_i)$	$Score(\mathbf{v}_i)$	$AccScore(\mathbf{v}_i)$
1	`00110101`	0.20703125	0.56235	0.56235	0.06560	0.06560
2	`10110010`	0.69531250	1.63441	1.63441	0.19067	0.25627
3	`10010111`	0.58984375	1.44967	1.44967	0.16911	0.42538
4	`10111011`	0.73046875	1.68896	1.68896	0.19703	0.62241
5	`10010111`	0.58984375	1.44967	1.44967	0.16911	0.79153
6	`11001110`	0.80078125	1.78708	1.78708	0.20848	1.00000

Ahora el individuo más apto es $\mathbf{v}_6 =$ `11001110` y el menos apto $\mathbf{v}_1 =$ `00110101`. El nuevo valor de la aptitud global es $Fitness(P[1]) = 8.57214980$.

Obsérvese la mejora predicha, no sólo en la aptitud global, sino también en la máxima y la mínima. Se terminará esta iteración para confirmar la tendencia.

2. Sorteo de individuos para reproducirse: Haciendo cálculos análogos a los de antes se obtiene

$$\begin{array}{lcl} 0.15816 & \rightarrow & \mathbf{v}_4 \\ 0.20190 & \rightarrow & \mathbf{v}_2 \\ 0.71533 & \rightarrow & \mathbf{v}_5 \\ 0.00178 & \rightarrow & \mathbf{v}_1 \\ 0.91161 & \rightarrow & \mathbf{v}_6 \\ 0.30624 & \rightarrow & \mathbf{v}_3 \end{array}$$

Ahora la selección no ha tenido ningún efecto sobre la población, por lo que se obtiene como población intermedia la misma del paso anterior (a pesar del desorden):

i	$\mathbf{v}'_i$
1	10111011
2	10110010
3	10010111
4	00110101
5	11001110
6	10010111

3. Selección de individuos para cruzarse. Los seis números aleatorios $r_i = \text{URand}[0,1)$, son ahora

0.27573   0.79741   0.56249   0.89852   0.21859   0.07933

por lo que los elegidos para el cruce serán $\mathbf{v}_1$, $\mathbf{v}_5$ y $\mathbf{v}_6$ de la nueva población. Como es una cantidad impar hay que elegir uno nuevo o descartar uno de ellos al azar.
En este caso resulta ser descartado el 6º, de modo que sólo se cruzarán $\mathbf{v}_1$ y $\mathbf{v}_5$

4. Cruce de los individuos anteriormente elegidos en un punto al azar. En este caso el punto de cruce resulta estar a la altura del 3^{er} bit.

$$\begin{array}{lcl} \texttt{101}|\texttt{11011} & & \mathbf{v}''_1 = \texttt{101}|\texttt{01110} \\ & \rightarrow & \\ \texttt{110}|\texttt{01110} & & \mathbf{v}''_5 = \texttt{110}|\texttt{11011} \end{array}$$

5. Los nuevos individuos sustituyen inmediatamente, a sus progenitores. De este modo la población intermedia tras el cruce es por lo que la población intermedia es ahora:

i	$\mathbf{v}''_i$
1	10101110
2	10110010
3	10010111
4	00110101
5	11011011
6	10010111

6. Selección de bits a mutar. Ahora resulta que no sale ningún bit para ser mutado, por lo que la población intermedia se mantiene tal cual.

7. Constitución de la nueva población. Que en este caso resulta ser la del quinto paso, ie.,

i	$\mathbf{v}_i[2]$
1	10101110
2	10110010
3	10010111
4	00110101
5	11011011
6	10010111

Si se volviesen a calcular los valores significativos se observaría una nueva mejora de las aptitudes.

Se ha resuelto este mismo problema en un computador utilizando el programa GENESIS de J.J. Grefenstette con los siguientes parámetros:

$$PopSize = 30 \quad Len = 12 \quad MaxIter = 50 \quad p_{xov} = 60\% \quad p_{mut} = 0.10\%$$

obteniéndose (ver apéndice A) la evolución que se muestra en la gráfica de la figura 2.5. Obsérvese cómo los mejores individuos están cerca del óptimo en muy pocas generaciones y la evolución exponencial de la aptitud media.

2.3 Análisis teórico de los AG

En los apartados anteriores se han centrado los esfuerzos en dejar claro *cómo* funcionan los AG. Se acometerá ahora la labor de explicar *por qué* funcionan los AG. En principio podría parecer que eso está fuera de lugar en una memoria de proyecto, pero no es este el caso ya que, con independencia de otras consideraciones, este análisis permitirá mostrar con claridad *qué* es exactamente lo que procesan los AG.

Efectivamente, un análisis superficial de lo visto hasta ahora lleva a pensar que los AGs procesan simplemente individuos, esto es, elementos codificados de

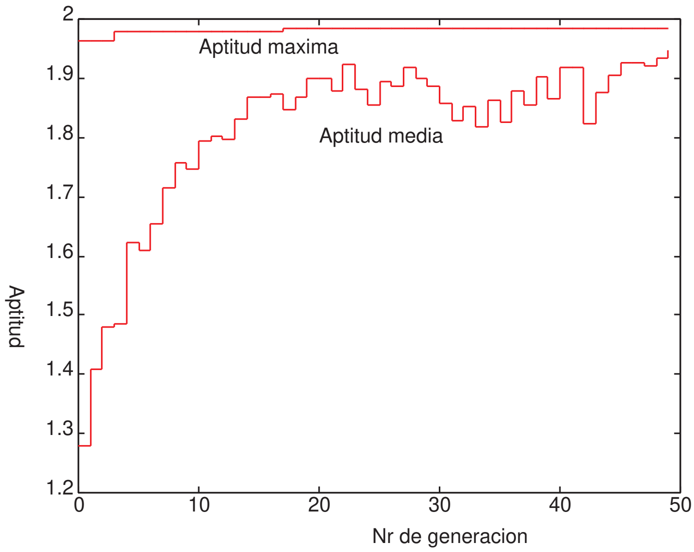

Figura 2.5: Evolución de las aptitudes

un espacio de búsqueda. Si ese fuera el caso, no tendría gran sentido la exigencia de codificación binaria; tomando sin ir más lejos el ejemplo de Riolo, el AG debería funcionar igual de bien con números en coma flotante que representasen directamente a las posibles soluciones, únicamente habría que modificar levemente los operadores genéticos. Si se plantea el problema con tal *representación directa* se puede ver que el AG resultante también converge, pero mucho más lentamente; de hecho no es más eficiente que otros métodos de búsqueda ciega. También con otros problemas se aprecia una fuerte pérdida de eficiencia del AG en comparación con el que usa la representación binaria.

La causa de esto radica en que los AGs no procesan estrictamente individuos, sino similitudes entre ellos, o mejor dicho *patrones de similitud* entre individuos, y dado que cada individuo encaja en muchos patrones a la vez, la eficiencia de la búsqueda se multiplica.

2.3.1 Definiciones y conceptos básicos

El fundamento teórico de los algoritmos genéticos reside en la noción de *esquema*.

Definición Un ***esquema*** es un patrón de similitud que se constuye introduciendo el signo "$*$" de *indiferencia* en el alfabeto de alelos. De este modo un esquema representa a todos los individuos que encajan en su representación.

Por ejemplo, el esquema `(*1*1100100)` representa a estas cuatro cadenas binarias:

`0101100100` `0111100100` `1101100100` `1111100100`

Naturalmente, el esquema (`1001110001`) sólo representa a una cadena y (`**********`) representa a todas las cadenas de longitud $Len = 10$.

Proposición Es inmediato verificar las siguientes propiedades:

1. Si un esquema contiene k símbolos de indiferencia entonces representa a 2^k cadenas binarias.

2. Una cadena binaria de longitud Len encaja en 2^{Len} esquemas distintos.

3. Considerando las cadenas binarias de longitud Len existen en total 3^{Len} posibles esquemas.

4. Una población de $PopSize$ cadenas binarias de longitud Len contiene entre 2^{Len} y $PopSize \cdot 2^{Len}$ esquemas distintos.

Para los propósitos que se señalaron resulta conveniente definir dos conceptos de gran importancia. Dado un esquema S se definen:

• **Orden de un esquema, o(S):** Es el número de posiciones especificadas (ie., con `0` ó `1`) que contiene dicho esquema.

• **Longitud característica de un esquema, δ(S):** Es la distancia en dígitos entre las posiciones fijas extremas.

Dado que un esquema S representa a $2^{Len-o(S)}$ cadenas, cuanto mayor sea el orden del esquema a menos cadenas representará. Por eso se dice que el orden de un esquema da una medida de su especificidad. En el presente desarrollo el orden servirá para calcular la probabilidad de supervivencia de un esquema frente a las mutaciones. Por otra parte, la longitud característica da una medida de la compacidad de la información contenida en el esquema. Nótese que un esquema con una sola posición especificada tiene una longitud característica de cero. Aquí se usará ese concepto para calcular la probabilidad de supervivencia de un esquema frente a los cruces.

Ejemplo: El esquema $S = (* * * * \mathtt{00} * * \mathtt{1} *)$ tiene un orden $o(S) = 3$ y una longitud característica $\delta(S) = 9 - 5 = 4$.

Dados una población de cadenas binarias $P[t]$ y un esquema S se definen también:

• **Presencia de S en P[t], ξ(S, P[t]):** Es el número de cadenas de $P[t]$ que encajan en el esquema S

• **Aptitud del esquema S en P[t], Fitness(S,P[t]):** Es el promedio de las aptitudes de todas las cadenas de la población que encajan en el esquema S

en el instante t. Esto es, si se numeran como $\mathbf{v}_1, \ldots, \mathbf{v}_p$, siendo $p := \xi(S,t)$, a las cadenas de $P[t]$ que encajan en S resulta que:

$$Fitness(S, P[t]) := \frac{1}{p} \sum_{i=1}^{p} Fitness(\mathbf{v}_i)$$

• **Aptitud media de la población en el instante t, AveFitn(P[t]):** Es el promedio de las aptitudes de todas las cadenas de la población en el instante t, o lo que es equivalente, la aptitud del esquema (*...*) en $P[t]$:

$$AveFitn(P[t]) := Fitness((*...*), P[t]) = \frac{1}{PopSize} \sum_{\mathbf{v}_i \in P[t]} Fitness(\mathbf{v}_i)$$

• **Aptitud relativa de S en P[t]:** Es el cociente entre la aptitud del esquema en la población y la aptitud media de la población en cierto instante, ie.,

$$AveFitn(S, P[t]) := \frac{Fitness(S, P[t])}{AveFitn(P[t])}$$

Partiendo de estas definiciones ya se puede comenzar el análisis teórico del funcionamiento de los AGs.

2.3.2 La ecuación de crecimiento de los esquemas

Se va a enunciar una ecuación de demostración sencilla que resultará esencial para comprender el modo de operación de los algoritmos genéticos. Con ella se muestra cómo evoluciona en promedio un esquema dentro de la población de un algoritmo genético.

Ecuación de crecimiento reproductivo de un esquema en una población: Sea $P[t]$ una población de $PopSize$ elementos de tamaño Len —con $PopSize$ lo suficientemente grande— que evoluciona a través de un AG clásico (SGA).
La presencia de un esquema S en la población $P[t]$ evoluciona en promedio según la siguiente fórmula

$$\boxed{\overline{\xi}(S, P[t+1]) = \overline{\xi}(S, P[t]) \cdot \kappa_g \cdot \kappa_s} \qquad [CR]$$

donde κ_g es el *factor de crecimiento* de S en la población y κ_s es el *factor de supervivencia*. El factor de crecimiento mide la tendencia del esquema a aumentar su presencia en la población intermedia y el factor de supervivencia mide la probabilidad de que pase a la siguiente generación. El elemento que hace aumentar la presencia de los esquemas en la población intermedia es la selección; de hecho es muy sencillo ver que, si se usa una selección por sorteo, la presencia de un esquema en la población intermedia $Q[t]$ evoluciona en promedio según:

$$\overline{\xi}(S, Q[t]) = \overline{\xi}(S, P[t]) \cdot AveFitn(S, P[t]) = \overline{\xi}(S, P[t]) \frac{Fitness(S, P[t])}{AveFitn(P[t])}$$

de manera que:

$$\kappa_g := AveFitn(S, P[t])$$

Ahora bien, existe la posibilidad de que los esquemas sean destruidos por los operadores genéticos antes de pasar a la próxima generación; es esta posibilidad la que cuantifica el factor de supervivencia κ_s. Tal factor es difícil de calcular exactamente, pero se puede acotar fácilmente mediante:

$$\kappa_s \geq \left(1 - \frac{p_{xov}\ \delta(S)}{Len - 1}\right) \cdot (1 - p_{mut})^{o(S)}$$

El primer factor mide la probabilidad de que S sobreviva a un cruce, o más exactamente, la probabilidad de que el cruce no ocurra sobre su longitud característica. El segundo factor mide la probabilidad de que no se vea afectado por una mutación. Todo esto presupone que se está usando el reemplazo inmediato.

En definitiva, siendo p_{xov} y p_{mut} las probabilidades de cruce y mutación (resp.) del SGA, la presencia de cierto esquema S en la población $P[t]$ evoluciona en promedio según:

$$\overline{\xi}(S, P[t+1]) \geq \overline{\xi}(S, t) \cdot AveFitn(S, P[t]) \cdot \left(1 - \frac{p_{xov}\ \delta(S)}{Len - 1}\right) \cdot (1 - p_{mut})^{o(S)}$$

Formulación aproximada: Como habitualmente $p_{mut} \ll 1$ se puede hacer

$$(1 - p_{mut})^{o(S)} \simeq 1 - p_{mut}\ o(S)$$

de modo que,

$$\kappa_s \gtrsim \left(1 - \frac{p_{xov}\ \delta(S)}{Len - 1} - p_{mut}\ o(S)\right)$$

lo que proporciona la siguiente formulación aproximada de la ecuación [CR]:

$$\overline{\xi}(S, P[t+1]) \geq \overline{\xi}(S, P[t]) \cdot AveFitn(S, P[t]) \cdot \left(1 - \frac{p_{xov}\ \delta(S)}{Len - 1} - p_{mut}\ o(S)\right)$$

2.3.3 El teorema fundamental de los algoritmos genéticos

Basándose en los resultados de la sección anterior cabe enunciar el siguiente teorema:

Teorema fundamental de los Algoritmos Genéticos: En las sucesivas generaciones de un SGA, la presencia de un esquema S en la población $P[t]$ evoluciona estadísticamente de modo exponencial según la ecuación [CR].

De la ecuación [CR] se extraen inmediatamente las siguientes consecuencias prácticas:

- La presencia de un esquema en una población evoluciona estadísticamente en progresión geométrica, progresión cuyo factor está determinado por la aptitud relativa del esquema, su longitud característica y su orden. Es fundamental darse cuenta de que los esquemas con una aptitud *por encima de la media* —abreviadamente, *esquemas aventajados*— incrementan exponencialmente su presencia en sucesivas generaciones ($\kappa_g > 1$), mientras que los que tienen la aptitud por debajo de la media decrementan exponencialmente su presencia en la población ($\kappa_g < 1$).

- La tendencia de los esquemas aventajados a incrementar su presencia en sucesivas generaciones se acentúa cuando el esquema es corto y de bajo orden, pues entonces $\kappa_s \simeq 1$.

Desde esta perspectiva, se puede explicar la introducción de los tres operadores del SGA de la siguiente manera:

1. *selección:* La selección se encarga de incrementar geométricamente la presencia de los esquemas aventajados y reducir la presencia de los retrasados. No obstante, la selección no introduce nuevos esquemas.

2. *cruce:* El cruce permite el intercambio estructurado de información útil (esquemas cortos, de bajo orden y aventajados) entre individuos. Así visto, el cruce es un operador esencial para el eficaz funcionamiento de un AG.

3. *mutación:* La mutación introduce variedad en el juego de esquemas de la población y proporciona un mecanismo de seguridad frente a posibles pérdidas de información valiosa. Visto de esta manera, la mutación es un operador secundario; de ahí que se aplique con bastante menor frecuencia que el cruce.

Existen buenos motivos para elegir una evolución geométrica para los componentes básicos de los AGs. La justificación de esta afirmación es prolija, baste con decir que así se proporciona la mejor tasa estadística de crecimiento de la aptitud media de la población para un número de iteraciones dado (ver [GOLD89,p36] y [KARG94]).

A la vista de todos estos resultados prácticos parece bastante razonable formular la siguiente hipótesis, que no ha sido completamente probada:

Hipótesis de los Bloques Constructivos: Los algoritmos genéticos exploran el espacio de búsqueda a través de la yuxtaposición de esquemas aventajados, cortos y de bajo orden. A tales esquemas se les llama *bloques constructivos*.

Así quedan respondidas las dos cuestiones formuladas al principio de esta sección, a saber:

1. ¿Por qué funcionan los AGs? Porque dan oportunidades de proliferación exponencialmente crecientes a los esquemas más aptos, y exponencialmente decrecientes a los menos aptos.
2. ¿Qué procesan los AGs? En sentido amplio los AGs procesan esquemas y en particular los AGs procesan de modo útil bloques constructivos.

Además queda justificada la exigencia del uso de representación binaria que se comentó al principio: la representación binaria es la que, para un tamaño dado, proporciona la máxima diversidad de esquemas.

Vista desde otra perspectiva, la hipótesis de los bloques constructivos afirma que cuanto más intervenga la construcción por bloques más eficaz será el funcionamiento del AG. Esa 'eficacia añadida' se pueden cuantificar a través del siguiente resultado, que no se demostrará:

Propiedad del Paralelismo Implícito de los AGs: Como ya quedó dicho, en una generación de $PopSize$ individuos binarios de tamaño Len existen entre 2^{Len} y $PopSize \cdot 2^{Len}$ esquemas distintos. Algunos de ellos, mayormente los bloques constructivos, serán procesados de modo útil, esto es, reproducidos exponencialmente y no alterados por los operadores genéticos. Pues bien, el mecanismo subyacente que da eficacia a los AGs a pesar de su generalidad se basa en el hecho de que, aunque el AG sólo procesa $PopSize$ estructuras en cada generación, se puede demostrar que, bajo hipótesis muy generales, se procesan de modo útil al menos $PopSize^3$ esquemas.

Es decir, si los AGs son más robustos que otros métodos de búsqueda ciega ello se debe a que realizan implícitamente una búsqueda en paralelo. Dicho paralelismo implícito se obtiene sin ningún requerimiento añadido de memoria o de potencia de cálculo, y seguramente sea el más destacado ejemplo de una explosión combinatoria que actúa de modo beneficioso.

Se pueden ilustrar estos resultados volviendo al ejemplo descrito en la sección anterior. Es muy sencillo ver que los esquemas más aventajados son `(*****100)` y

(11111***). El primer esquema representa a los puntos más altos de cada pico y el segundo representa a todos los puntos del último pico. Si la población inicial es lo suficientemente variada, en las primeras etapas el AG favorecerá la proliferación de los individuos que contengan el primer esquema, después dará preferencia a aquéllos que se ajusten más al segundo, hasta finalmente dar con el punto óptimo 11111100, que combina ambos.

2.3.4 Limitaciones prácticas del análisis teórico

Todos los resultados anteriores están basados en en la ecuación [CR] la cual presupone tres requisitos:

1. Por motivos técnicos, se debe imponer que la función de aptitud devuelva siempre valores no negativos.

2. Se admite que el número de generaciones del AG es infinito. Como esto no es viable, se supone un número de generaciones lo suficientemente grande como para que las 'magnitudes esperadas' sean útiles en la práctica.

3. Se admite que el número de individuos de la población es infinito. Esto tampoco es viable, así que se supone que los operadores funcionan de manera razonablemente insesgada con respecto a los valores esperados y que *PopSize* es lo suficientemente grande como para mantener en todo momento una *población variada*.

Para solventar el primer requisito se distingue la función de aptitud de la función de evaluación a través de un reescalado, como luego se verá. El segundo requisito se soluciona aceptablemente en la práctica ejecutando el AG varias veces para resolver un problema. El último requisito es bastante más problemático y merece ser estudiado con detenimiento, cosa que se hará en el apartado 3.1.2; baste de momento la siguiente justificación: por su propia definición, para una población infinita en todo momento es posible encontrar cierto individuo o cierto esquema; de un individuo a otro sólo varía la mayor o menor probabilidad de aparición. Sin embargo, en una población finita siempre existe el riesgo de que desaparezca de la población, por cuenta del proceso de selección, un individuo que puediera ser importante en posteriores etapas, siendo ésto tanto más probable cuanto más sesgado esté el proceso de selección hacia los más aptos. Esto perjudica de modo más o menos serio a la variedad de la población, lo que origina una cascada de problemas en el funcionamiento del AG. Por si no fuera bastante, al usar poblaciones finitas es inevitable introducir errores estadísticos en la aplicación de los operadores, especialmente en los muestreos. Si no se corrigen, esos errores agravan más todavía el problema de la pérdida de la diversidad.

Aún queda sin considerar otra limitación que surge en la práctica: Se ha supuesto implícitamente que la codificación es perfecta; esto conlleva, entre otras cosas, que a cada punto del dominio del problema le corresponde uno y sólo un individuo codificado en el espacio de búsqueda. No siempre es este el caso, por lo que se deberán habilitar procedimientos auxiliares de discriminación que de un modo u otro perjudicarán a la eficiencia del AG.

Capítulo 3

Dificultades en el uso de los AGs. Planteamiento y resolución

A la hora de llevar a la práctica el modelo de los Algoritmos Genéticos para resolver problemas no triviales de optimización se plantean varias dificultades de orden teórico y práctico. Todas ellas surgen como consecuencia de cuatro limitaciones intrínsecas de dicho modelo; por eso, para corregir o mitigar sus efectos conviene estudiar antes con cierto detenimiento las implicaciones generales de tales limitaciones. Sólo entonces se podrá acometer el diseño de procedimientos eficaces de resolución.

3.1 Limitaciones de los AGs e inconvenientes asociados

De modo general, las cuatro limitaciones principales del AG básico (SGA) son:

1. *La representación:* El espacio de búsqueda sólo puede ser representado mediante cadenas binarias.

2. *La irrestricción:* El mecanismo del SGA no considera las posibles de restricciones o ligaduras que pudieran imponerse a la búsqueda.

3. *La opacidad:* El SGA es un algoritmo de búsqueda ciega, esto es, sólo está guiado por la aptitud de los individuos, y no incorpora ningún otro conocimiento específico del problema en cuestión.

4. *La finitud:* A efectos prácticos, el SGA sólo puede trabajar con poblaciones finitas no muy grandes.

Los inconvenientes que ocasionan las dos primeras limitaciones no se salen de lo previsible; por contra, la opacidad y la finitud son causa original de otros muchos inconvenientes que habrá que atacar por separado. De manera esquemática, pude decirse que todos esos inconvenientes tienen su base en dos hechos:

1. Fundamentalmente, la opacidad es causa inevitable de *debilidad*.
2. Por otra parte, la opacidad acentúa fuertemente los *problemas de diversidad* derivados de la finitud de la población.

Seguidamente se estudiarán más detenidamente sus consecuencias.

3.1.1 El problema de la debilidad de los AGs

El inconveniente más visible de la debilidad es la ***ineficiencia***: los métodos débiles de resolución de problemas presentan las ventajas de ser muy poco específicos, poco exigentes y notablemente eficaces; sin embargo, todos ellos son ineficientes en mayor o menor grado, y especialmente si se les compara con métodos heurísticos. Los AGs sólo se pueden considerar eficientes en comparación con otros métodos estocásticos de búsqueda ciega, pero se puede garantizar que si se encuentra un método heurístico para resolver un problema este siempre lo hará mucho más eficientemente que un AG.

Pero eso no es todo: otro de los inconvenientes prácticos de la debilidad es la tendencia al ***extravío*** de la búsqueda. El fenómeno es así: el SGA realiza la búsqueda de los mejores puntos utilizando únicamente la aptitud de los individuos para recombinar internamente los bloques constructivos; en ocasiones ocurre que la información proporcionada (ie., la aptitud) resulta insuficiente para orientar correctamente al algoritmo en su búsqueda del óptimo. A esto se le llama ***desorientación*** (en la bibliografía *deception*). A nivel interno esto se concreta en que no se verifica la hipótesis de los bloques constructivos, es decir, ciertas combinaciones válidas de buenos bloques constructivos originan individuos de baja aptitud. En el peor de los casos puede ocurrir que este fenómeno tenga tanta fuerza como para impedir que el AG converja al óptimo global, desviándolo finalmente hacia un óptimo local. Esto es el extravío propiamente dicho. Afortunadamente, aunque un AG se desoriente no necesariamente se va a extraviar; para eso es necesario partir de una mala población inicial o plantear un problema especialmente diseñado para que se extravíe.

En resumen, para que funcione correctamente el mecanismo básico de los AGs —la construcción por bloques— es necesario proporcionarle una mínima cantidad (y calidad) de información. Si no se le proporciona ese mínimo el mecanismo no funciona con propiedad, y el AG evolucionará incorrectamente.

Como es de suponer, el factor que más contribuye a la existencia de desorientación es la forma de la función de aptitud. Típicamente, la desorientación es frecuente en problemas en los que los puntos óptimos están aislados y rodeados de puntos de muy baja aptitud. Sin embargo, conviene señalar que la presencia de desorientación suele estar fuertemente estimulada por una mala codificación que 'oculte' la ya de por sí escasa información disponible. A nivel interno esto se explica por la mayor dificultad para constituir los bloques constructivos. De modo recíproco, una buena codificación reduce la probabilidad de extravío (ver también apdo. 3.2.1.2).

Un caso especialmente desfavorable ocurre cuando hay una fuerte interacción entre dos o más atributos (ie., genes), de tal forma que la contribución a la aptitud de un individuo que realiza cierto gen depende grandemente de los valores que tomen otros. A este fenómeno se le dice ***epistasis*** o *acoplamiento* y su presencia garantiza la desorientación dado que en esas condiciones resulta muy difícil constituir buenos bloques constructivos.

En la sección 3.2 se verán algunas maneras de atenuar los efectos de los inconvenientes asociados a la debilidad. De todos modos, hay ocasiones en que estos remedios resultan insuficientes y no cabe más solución que eliminar la debilidad incorporando conocimiento específico al AG; eso será materia del próximo capítulo.

3.1.2 El problema de la diversidad en los AGs

A efectos prácticos, es fundamental que todo AG verifique lo siguiente

> ***Requisito de variedad en los AGs:*** Es esencial para el buen funcionamiento de un AG tener controlada en todo momento la diversidad de la población. Se entiende la diversidad en sentido general como 'diversidad de individuos' y en particular como 'diversidad de aptitudes'.

La necesidad de que haya diversidad de individuos es fácil de entender: con poca variedad de individuos hay poca variedad de esquemas, a causa de ello el operador de cruce pierde casi por completo la capacidad de intercambio de información útil entre individuos, y en definitiva, la búsqueda se estanca. La necesidad de tener controlada la diversidad de aptitudes radica en la imposibilidad práctica de trabajar con una población infinita, como enseguida se verá.

Es fácil darse cuenta de que no es conveniente tener poca diversidad de aptitudes ya que en tal caso todos los individuos tendrían más o menos las mismas posibilidades de sobrevivir, la selección reproduciría la situación anterior y todo el peso de la búsqueda recaería en los operadores genéticos, lo que a la larga sería poco más que una búsqueda aleatoria. El resultado final es que la búsqueda se estanca y, como luego se verá, la situación puede empeorar si la población es finita. En definitiva, para que la selección sea efectiva, la población debe contener en todo momento una cierta variedad de aptitudes.

Por otra parte, cuando la población es finita —es decir, siempre— tampoco se puede tener gran disparidad de aptitudes, pues ello suele afectar muy negativamente a la diversidad de la población, como seguidamente se explica.

3.1.2.1 Convergencia prematura por problemas con la diversidad

Sea un AG básico con población finita y admítase por simplicidad que la función de aptitud coincide con la función de evaluación. En algún momento puede ocurrir que un individuo o un grupo de ellos obtengan una aptitud notablemente superior a los demás. Esto es especialmente probable en las fases tempranas de la evolución, en las cuales de entre una población de individuos mediocres suele surgir un buen candidato por aplicación de los operadores genéticos. En tal circunstancia existe el riesgo de que se produzca una *evolución en avalancha*: al incrementar los individuos más aptos su presencia en la población, por ser ésta finita la diversidad disminuye, ello hace que en la siguiente generación se favorezca aún más a los individuos más aptos hasta que éstos acaban dominando por completo la población. A esto se le conoce como el fenómeno de los ***superindividuos***. Habitualmente ocurre que tales superindividuos sólo son los más aptos en cierto momento, pero no los más aptos absolutos —téngase en cuenta que la falta de diversidad estanca la búsqueda— lo que en último término provoca una ***convergencia prematura*** del AG, habitualmente hacia un subóptimo.

La única circunstancia en que este fenómeno es tolerable e incluso deseable es en las fases tardías de la evolución, cuando el AG ha 'localizado' correctamente la solución óptima, pero nunca antes.

La posibilidad de convergencia en avalancha es un fenómeno consustancial a los AGs con población finita, debido que la finitud activa el bucle 'selección sesgada $\rightleftharpoons$incremento de la diversidad'. Tanto es así que dicho fenómeno puede ocurrir incluso cuando no haya diferencias apreciables en las aptitudes de la población. Ya se comentó que en tal circunstancia la búsqueda se estanca, dado que la selección no concede mayor preeminencia a uno u otro individuo. Sin embargo, cuando la población es finita y especialmente cuando no es muy grande, puede ocurrir el siguiente fenómeno, conocido como ***deriva genética***: En una población finita los procesos de muestreo son siempre imperfectos, pudiendo favorecer más de lo que les corresponde a individuos ocasionalmente más aptos; debido a la finitud y como consecuencia de esos errores estocásticos en los muestreos, un individuo puede acabar 'expulsando' de la población a los restantes, haciendo que el AG converja prematuramente hacia tal 'individuo afortunado'.

Dado que en una población finita son inevitables los ruidos estocásticos introducidos por el muestro, el peligro de deriva genética está siempre presente; en la práctica sólo es importante cuando las aptitudes son todas parecidas (poca diversidad de aptitudes) o con pequeños tamaños de población.

En resumen, es imprescindible tener control sobre la diversidad de aptitudes de la población (finita) para evitar que se produzca una convergencia prematura (avalancha) ya sea por mucha diversidad (superindividuos) o incluso por demasiado poca (deriva genética).

3.1.2.2 Mecanismos de control de la diversidad. Presión selectiva

Para controlar la diversidad de las aptitudes se debe actuar sobre el mecanismo de asignación de probabilidades de supervivencia (o de descendientes, en su caso): cuanto más se favorezca a los más aptos menor variedad se obtendrá, y como consecuencia aumentará el riesgo de perder la información de valor desarrollada por los individuos moderadamente aptos, entendiendo por 'información de valor' toda la que podría servir posteriormente para hallar el óptimo global. Por el contrario, si no se favorece especialmente a los individuos más aptos, se obtendrá más diversidad en la población, pero a costa de hacer las etapas de selección mucho menos eficaces, lo que empeora la eficacia global del AG.

A la mayor o menor tendencia del mecanismo de asignación de probabilidades de supervivencia a favorecer a los individuos más aptos se le llama ***presión selectiva***. Lo ideal sería que en las primeras fases de la evolución de un AG hubiera poca presión selectiva con el fin de que la búsqueda fuera lo más global posible y en las últimas fases de la evolución, una vez que ya ha sido 'localizada' la mejor solución, incrementara fuertemente la presión selectiva para encontrarla cuanto antes.

Existen varias formas de cuantificar el concepto de presión selectiva en un valor numérico; la más sencilla posiblemente sea:

$$SelPres[t] := PopSize \cdot MaxScore[t]$$

Para mantener una distribución de aptitudes razonablemente uniforme, Whitley [PROC89,p116] recomienda trabajar con valores de $SelPres[t]$ lo más próximos posible a 1.5; valores mayores ocasionarían superindividuos, y valores menores frenarían la búsqueda sin ningún beneficio adicional. Evidentemente, el mínimo valor es $SelPres = 1$, que corresponde a la situación en que todos los individuos son iguales (diversidad nula).
Obsérvese que esa definición es equivalente a:

$$SelPres[t] \equiv \frac{MaxFitn[t]}{AveFitn[t]}$$

De acuerdo con la ecuación [CR] eso significa que el parámetro $SelPres$ no es más que una manera de cuantificar el número esperado de descendientes que se dan al miembro más apto de la población.

En lo que respecta al concepto de diversidad, se suele concretar en un número a través de la dispersión típica de las puntuaciones:

$$Div[t] := \sqrt{\sum_{\mathbf{v} \in P[t]} \left(Score(\mathbf{v}) - AveScore[t]\right)^2}$$

donde, como ya se sabe, $Score(\mathbf{v}) := Fitn(\mathbf{v}) / \sum_{\mathbf{v} \in P} Fitn(\mathbf{v})$.

Si todos los individuos tienen la misma aptitud la diversidad será cero, y si las aptitudes están uniformemente repartidas entre $MinFitn$ y $MaxFitn$ la diversidad valdrá $(MaxScore - MinScore)/2\sqrt{3}$, esto es, $(1/2\sqrt{3}) \cdot (MaxFitn - MinFitn)/AveFitn$; valores apreciablemente mayores señalan la presencia de superindividuos. Evidentemente, dado que $\sum_{\mathbf{v}\in P} Score(\mathbf{v}) = 1$ un valor alto de $SelPres$ implicará un bajo valor de Div, lo cual viene a confirmar el hecho ya conocido de que diversidad y presión selectiva son factores en conflicto.

A efectos prácticos, se puede controlar la diversidad de aptitudes de cuatro maneras:

1. Actuando directamente sobre el mecanismo de selección: Se trata de hacerle lo más insensible posible a la distribución de aptitudes y a los errores de muestreo, con la idea de evitar la convergencia prematura. Se consigue usando mecanismos de muestreo distintos del de por sorteo.

2. Actuando sobre el mecanismo de asignación de aptitudes: Se trata de conseguir una distribución de aptitudes lo más uniforme posible modificando apropiadamente el mecanismo de asignación de aptitudes. Así la probabilidad de supervivencia deja de ser necesariamente proporcional a la aptitud bruta.

3. Controlando las 'edades' de los individuos.

4. Introduciendo mecanismos de escisión.

Cada uno de estos procedimientos se estudiará a su debido tiempo.

No se debe confundir la presión selectiva con el grado de penetración. El grado de penetración mide la preeminencia que se da a la búsqueda en profundidad (exploración) sobre la búsqueda en anchura (explotación), mientras que la presión selectiva mide la preeminencia que se da a la selección sobre la búsqueda propiamente dicha, y sólo tiene sentido para métodos de búsqueda multiple. En la figura 3.1 se muestra cómo se agrupan los candidatos a solución en torno a un subóptimo dependiendo de los valores que se den a la presión selectiva y al grado de penetración.

3.2 Resolución de inconvenientes, mejoras y alternativas

Se van a describir algunas modificaciones del modelo básico de algoritmos genéticos. Se mantendrá punto por punto la estructura fundamental, introduciendo simplemente variantes y parametrizaciones en las configuraciones por defecto de los nueve métodos y procedimientos indicados en 2.1.1.

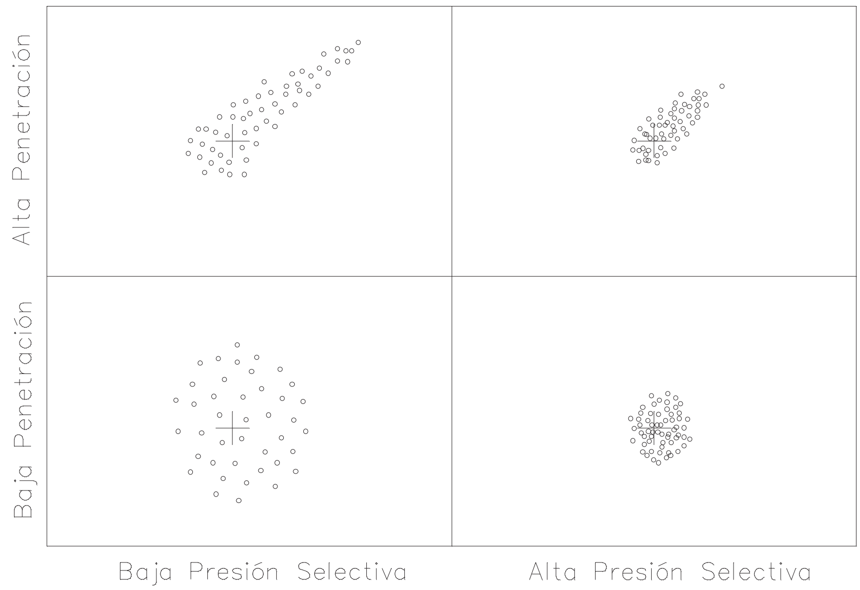

Figura 3.1: Presión selectiva vs. grado de penetración

3.2.1 El problema de la representación y la codificación

Como ya quedó dicho, los AGs procesan esquemas, esto es, patrones de similitud. Por ello, con la idea de hacer la búsqueda más eficiente, se representan los puntos del espacio de búsqueda mediante cadenas binarias pues esta es la representación que proporciona la mayor variedad de esquemas para una longitud dada. Esto plantea graves inconvenientes que se resumen en lo siguiente: la representación binaria no resulta *útil* para la mayoría de problemas de búsqueda en el sentido de que no siempre es posible encontrar una codificación que dé significado a cada una de las posiciones de que consta el genotipo; es más, aunque se encuentre, eso no garantiza que represente con fidelidad el dominio del problema. Ambas cuestiones se estudiarán por separado.

Comenzando por la segunda cuestión —la representación con fidelidad del dominio del problema—, es conveniente introducir la siguiente definición: Se dice que una representación es ***perfecta*** desde el punto de vista de los objetos representados cuando cumple estas cinco propiedades:

1. *Completitud:* Debe poder representar todos los objetos de la clase indicada.

2. *Coherencia:* Únicamente debe representar objetos de esa clase.

3. *Uniformidad:* Todos los objetos deben estar representados por la misma cantidad de codificaciones, ie., no debe haber objetos 'sobrerrepresentados' ni 'infrarrepresentados'.

4. *Sencillez:* Debe ser fácil de aplicar el mecanismo de codificación objeto$\leftrightarrow$ individuo tanto en sentido directo como inverso.

5. *Localidad:* Pequeños cambios en los individuos se han de corresponder con pequeños cambios en los objetos.

En la práctica suele ser imposible encontrar una *representación perfecta* del dominio del problema a través de cadenas binarias, lo cual introduce una diferencia más o menos grande entre lo que se está buscando y lo que se quiere buscar.

La otra cuestión, la de la utilidad de la representación, no es trivial: es muy importante que las Len posiciones de que consta el fenotipo tengan algún significado concreto; de este modo, si se ordenan apropiadamente dichas posiciones, los bloques constructivos serán cortos y compactos, lo que dará eficacia a la búsqueda genética. De hecho, es tan necesario que las regiones de gran aptitud se representen mediante esquemas cortos y compactos que en ocasiones se sacrifica el requisito de representación binaria si ello conlleva mayor naturalidad. Nótese que ésta es la única vía por la que se puede introducir conocimiento específico en un AG. En la práctica, la ordenación más común consiste en agrupar las posiciones por atributos (ie., genes) y concatenar los grupos a la manera que se indicó en el apdo. 2.1.3

Toda esta discusión se sintetiza en la siguiente recomendación:

> Se aconseja siempre realizar la elección de la representación basándose en estos dos principios:
> *I. Principio de los bloques constructivos con significado*: Las zonas del espacio de búsqueda 'relevantes para el problema' deben poderse representar mediante esquemas cortos y de bajo orden (bloques constructivos).
> *II. Principio del alfabeto de símbolos mínimo*: Se debe elegir el alfabeto de símbolos más pequeño (preferentemente binario) que permita una expresión natural del problema.

Se ilustrarán ambas ideas sobre dos ejemplos.

El problema binario de la mochila: Se trata de elegir un grupo de objetos de entre un conjunto de m con volúmenes $w_1, \ldots, w_m$ y valores $p_1, \ldots, p_m$ de manera tal que quepan en una mochila de capacidad C maximizando el valor total de su contenido.

Este problema admite múltiples formulaciones según se dé uno u otro significado a las variables w_i y p_i. Matemáticamente, se puede plantear como un problema de programación binaria así:

Encontrar para un vector de valores $\mathbf{p} := (p_1, \ldots, p_m)$ y otro de volúmenes $\mathbf{w} := (w_1, \ldots, w_m)$ un vector binario de selección $\mathbf{x} := (x_1, \ldots, x_m)$ con el que $x_j = 1$ indica que se ha seleccionado el j-ésimo objeto, de modo que haga máximo el valor total $f(\mathbf{x}) = \mathbf{p}' \cdot \mathbf{x}$ sujeto a la condición de capacidad limitada $\mathbf{w}' \cdot \mathbf{x} \leq C$, es decir:

$$\left\{ \begin{array}{rrcl} \text{maximizar} & f(\mathbf{x}) & = & \mathbf{p}' \cdot \mathbf{x} \\ \text{sujeto a} & \mathbf{w}' \cdot \mathbf{x} & \leq & C \\ \text{siendo} & x_i & \in & \{0, 1\} \quad (\forall i = 1, \ldots, m) \end{array} \right.$$

Se define por comodidad el vector de valores específicos $\boldsymbol{\rho} = (\rho_1, \ldots, \rho_m)$ a través de $\rho_i := p_i / w_i \quad (\forall i = 1, \ldots, m)$.

La representación de este problema para resolverlo mediante un AG es elemental: los individuos $\mathbf{v}$ del espacio de búsqueda coinciden con los puntos $\mathbf{x}$ del dominio del problema. La j-ésima posición del genotipo señala la presencia o ausencia de j-ésimo objeto en la mochila. Esta representación es la más sencilla y también la más natural dado que los genes tienen un significado claro e inmediato. Además, si se ignora la restricción de capacidad (espacio de búsqueda irrestricto) existe una correspondencia biunívoca entre los individuos y los puntos del dominio, por lo que la representación es perfecta.

Desafortunadamente, es esencial considerar la restricción en este caso, pues al eliminarla el problema se trivializa. Como próximamente se verá (apdo. 3.2.2), para poder tener en cuenta la restricción conviene representar los candidatos a solución del problema de modo indirecto, de tal suerte que se perderá en gran medida la naturalidad y la perfección de la anterior representación.

El problema de optimización paramétrica: Se engloban en esta denominación a todos los problemas de programación estática uniobjetivo en el dominio real, los cuales responden a la formulación

$$\left\{ \begin{array}{rl} \text{maximizar} & f(\mathbf{x}) \\ \text{sujeto a} & \mathbf{x} \in \mathcal{X} \subseteq \mathcal{R}^m \end{array} \right.$$

donde $\mathbf{x}$ es el vector de parámetros.

Para este problema no existe una representación natural en sentido estricto; es decir, en la representación concatenada habitual cada parámetro se codifica en un gen como un entero binario, pero no se puede dar un significado útil a cada una de las posiciones. Repárese, sin ir más lejos, en el ejemplo de Riolo ¿Qué significado práctico tiene cada uno de los ocho bits con los que se codificó el intervalo $[0, 1]$? Muy poco, y aunque se eligiera otra codificación seguirían sin tener gran significado práctico. Esto conllevará la dificultad de constituir bloques constructivos con significado, salvo que haya algún tipo de regularidad en el problema que haga posible la constitución de tales bloques, como ocurría en el ejemplo de Riolo. De todos modos esto no se puede esperar en un problema genérico.

El problema de la optimización paramétrica lleva también a plantearse cual es la mejor manera de discretizar y codificar el dominio $\mathcal{X}$. De eso tratará el próximo apartado.

3.2.1.1 Discretización y codificación de dominios continuos

Por construcción los AGs sólo pueden operar en dominios discretos, por eso cuando el dominio del problema es continuo se le debe discretizar previamente. Para problemas de optimización paramétrica la discretización es relativamente sencilla: para determinar el espacio de búsqueda X se eligen dos vectores $\mathbf{l}$ y $\mathbf{u}$ tales que el dominio $\mathcal{X}$ esté contenido en el paralelepípedo determinado por esos dos vectores, ie.,

$$\mathcal{X} \subseteq X := [l_1, u_1] \times [l_2, u_2] \times \ldots \times [l_m, u_m]$$

Entonces, para una tolerancia TOL cada parámetro x_j se codifica en L_j bits, donde

$$L_j = \left\lceil \log_2 \left(1 + \frac{u_j - l_j}{TOL} \right) \right\rceil \qquad (\forall j = 1, \ldots, m)$$

como ya se sabe.

Además se debe dotar al AG con un procedimiento de consideración de restricciones, dado que no todos los individuos del espacio de búsqueda, X, representan a soluciones factibles —ie, dentro de $\mathcal{X}$.

El problema de esta metodología reside en que no siempre es fácil determinar los valores más apropiados para los vectores de cotas $\mathbf{l}$ y $\mathbf{u}$, pues habitualmente $\mathcal{X}$ viene descrito implícitamente a través de un sistema de inecuaciones. Con los recursos de que se dispone hasta el momento ese inconveniente no tiene una solución sencilla: no es conveniente acotar por exceso, pues ello produciría individuos muy largos que consumirían muchos recursos de computación y además el AG ocuparía la mayor parte del tiempo en procesar puntos no factibles.

Otro problema reside en el hecho de que con la codificación binaria habitual no se cumple el requisito de localidad, de modo que puntos muy próximos tienen codificaciones muy distintas. Eso es así porque con frecuencia dos enteros binarios consecutivos coinciden en muy pocos de sus respectivos dígitos —esto se describe gráficamente con el nombre de *acantilados de Hamming*—. Como consecuencia de esto, una mutación en un individuo puede producir cambios drásticos en los puntos representados.

Se puede corregir en gran medida este problema utilizando otras codificaciones binarias que conserven la propiedad de adyacencia, es decir, que codifiquen enteros consecutivos como cadenas binarias que difieren en un solo bit. La más conocida de estas codificaciones es la de Gray.

La ***codificación de Gray*** asocia a cada entero de la sucesión $0, 1, \ldots, 2^N - 1$ una cadena de bits de longitud N que se construye aplicando sucesivamente el siguiente procedimiento:

> Comenzando por la cadena `0`...`0`, en cada iteración se conmuta el bit menos significativo que produzca una nueva cadena.

Por ejemplo, la sucesión $0, 1, \ldots, 7 = 2^3 - 1$ tiene la siguiente codificación de Gray:

`000   001   011   010   110   111   101   100`

Para implantarlo en un computador se programan estas dos rutinas de conversión de codificación natural a codificación de Gray y viceversa:

```
BeginAlgorithm{Bin2Gray}
   for( i = N; i >= 0; i--)
      GRA[i] = BIN[i+1] xor BIN[i];
EndAlgorithm

BeginAlgorithm{Gray2Bin}
   for( i = N; i >= 0; i--)
      BIN[i] = BIN[i+1] xor GRA[i];
EndAlgorithm
```

donde $i = N$ indica el bit más significativo de la cadena e $i = 1$ el bit menos significativo.

El uso de la codificación de Gray reduce grandemente la frecuencia de aparición de los acantilados de Hamming, y cuando aparecen tienen más amplitud. Ambos hechos benefician al funcionamiento de los AGs, pues interesa que la mayoría de veces que se apliquen las mutaciones proporcionen ligeros cambios y unas pocas veces originen cambios drásticos ya que es así como se puede dar con una zona de exploración completamente nueva.

Discretización de problemas de programación dinámica: En los problemas de programación dinámica los objetos buscados son *funcionales*, esto es, aplicaciones sobre un dominio real. La manera más sencilla de discretizar los funcionales es a través de listas de puntos, pero habitualmente resulta preferible discretizarlos a través de β-esplines y codificarlos indirectamente mediante los parámetros de dichos β-esplines.

3.2.1.2 Mejora evolutiva de la codificación. El operador de inversión

Es muy importante que las regiones del espacio de búsqueda con gran aptitud se puedan representar mediante esquemas cortos y compactos (principio de los bloques constructivos con significado). En caso de que esto no ocurra, el AG tendrá dificultades para converger y, como ya se comentó en 3.1.2, es bastante probable que se desoriente y acabe extraviándose.

Existe una manera bastante eficaz de estimular la formación de buenos bloques constructivos sometiendo a la propia distribución de posiciones al proceso de evolución. Ello se consigue introduciendo un nuevo operador genético llamado ***inversión*** o también *mutación estructural*:
La inversión es un operador unitario que selecciona dos puntos en una cadena e invierte el orden de las posiciones intermedias recordando el significado de cada una. Esto último obliga a identificar los bits en las cadenas, lo cual se lleva a cabo colocando a cada alelo una etiqueta con su posición.

Por ejemplo, al invertir la cadena

$$\langle 0_1 0_2 0_3 | 1_4 1_5 0_6 1_7 | 0_8 0_9 0_{10} 1_{11} \rangle$$

entre los puntos señalados se obtiene

$$\langle 0_1 0_2 0_3 | 1_7 0_6 1_5 1_4 | 0_8 0_9 0_{10} 1_{11} \rangle$$

Nótese que el contenido de la cadena no se altera, tan sólo se reorganiza su estructura; dicho de otro modo, tras la inversión la cadena sigue representando al mismo individuo, sólo que con otra estructura. De ahí el nombre de mutación estructural.

Obsérvese que la incorporación de etiquetas hace que no siempre sea posible aplicar correctamente el operador cruce, por lo que habrá que habilitar criterios con los que tratar los cruces erróneos. Se volverá sobre este tema en la sección 5.2.

Se introduce este operador con la idea de que el AG busque no sólo el mejor fenotipo (contenido) sino el mejor genotipo (estructura). Esto es especialmente útil cuando está bloqueado el mecanismo de formación de bloques constructivos, pues muchas veces basta con introducir este operador para desbloquearlo.

De todos modos, para muchos problemas que surgen en la práctica la representación mediante cadenas binarias es demasiado pobre, de manera que es imposible encontrar una ordenación de las posiciones que verifique mínimamente el principio de los bloques constructivos con significado. En tal caso la inversión proporciona una ayuda limitada y posiblemente haya que recurrir a otras opciones tales como incorporar conocimiento específico o bien utilizar una variedad especialmente diseñada para resolver este tipo de problemas, los *AGs tramados*. La incorporación de conocimiento específico en los AGs se estudia detenidamente en el apartado 3.2.9 y a lo largo de todo el capítulo 3. En cuanto a los AGs tramados, se describen con detalle en la referencia [GOLD93] de la bibliografía y se implantan en el paquete informático `mGAinC`, documentado en [DEB93].

3.2.2 Tratamiento de los puntos no factibles

Como ya quedó dicho en 3.1 una de las limitaciones del AG básico es que carece de mecanismos para considerar las posibles restricciones o ligaduras que pudieran imponerse a la búsqueda; es decir, el SGA es esencialmente un mecanismo de *búsqueda irrestricta*. Ya se anticipó en el apdo. 3.2.1 cuál era el problema en concreto: para poder usar la representación binaria se debe codificar sobre un superconjunto del dominio del problema $X \supseteq \mathcal{X}$. La elección de tal superconjunto está condicionada por dos factores

1º. Debe proporcionar una fácil codificación
2º. El superconjunto X no debería ser mucho más grande que el dominio $\mathcal{X}$, pues eso obligaría al AG a gastar la mayor parte de su tiempo en procesar puntos no factibles.

En la práctica dicha elección es notablemente discrecional.

El verdadero problema surge porque, dada la rigidez de la representación, no se puede esperar que el espacio de búsqueda del AG, X, coincida con el dominio del problema $\mathcal{X}$; razonablemente existirán en el espacio de búsqueda puntos no factibles que habrá que tratar de alguna manera.

Tal y como se plantea el AG básico, éste es un método de búsqueda intrínsecamente irrestricto y cualquier remedio pasará por incorporar, de modo directo o indirecto, conocimiento específico del problema.

Tradicionalmente se han empleado tres tipos de técnicas para resolver este problema, cada una con sus ventajas e inconvenientes:

Técnicas de penalización: Consisten en resolver directamente el problema irrestricto, penalizando a los individuos no factibles. En los AGs eso se lleva a cabo generando individuos sin tener en cuenta su factibilidad reduciendo la aptitud de los que resulten ser no factibles.
La única exigencia de estas técnicas es que debe poderse medir de alguna manera el 'grado de no-factibilidad', para así dar mayor penalización a los puntos más alejados del dominio del problema. Ésto es fácil de hacer en problemas de optimización paramétrica, pues en ellos se suele definir $\mathcal{X}$ implícitamente a través de un conjunto de p restricciones expresadas como inecuaciones así

$$g_i(\mathbf{x}) \geq 0$$

En tal circunstancia es inmediato definir el 'grado de violación de la i-ésima restricción' de este modo,

$$h_i(\mathbf{x}) = \begin{cases} -g_i(\mathbf{x}) & \text{sii} \quad g_i(\mathbf{x}) < 0 \\ 0 & \text{sii} \quad g_i(\mathbf{x}) \geq 0 \end{cases} \qquad (\forall i = 1, \ldots, p)$$

e incluir esos h_i en la función de evaluación así:

$$Eval(\mathbf{v}) \leftarrow Eval(\mathbf{v}) - K \cdot Penalty(h_1(\mathbf{v}), \ldots, h_p(\mathbf{v}))$$

donde p es el número total de restricciones
h_i es el grado de violación de la i-ésima restricción
K es un coeficiente global de penalización

La función $Penalty(\cdot, \ldots, \cdot)$ es una función positiva [1] y creciente que se define *ad hoc* para cada problema. Bajo ciertas condiciones de 'buen comportamiento' se puede demostrar que la solución del problema irrestricto converge a la del problema original restringido cuando $K \to \infty$ independientemente de la forma en particular que tenga la función de penalización.

Por ejemplo, Michalewicz [MICH94,p134] propone la siguiente función de penalización para resolver problemas típicos de optimización paramétrica a través de un AG:

$$\text{Penalty}(\mathbf{x}[t]) = k \left(\frac{t}{MaxIter}\right)^r \text{AveEval}(P[t]) \sum_{i=1}^{NrRes} h_i(\mathbf{x}[t])$$

donde $NrRes$ es el número de restricciones
$MaxIter$ es el número máximo de iteraciones a realizar
$k \simeq 1/NrRes$
$r \simeq 1$
h_i es el grado de violación de la restricción i-ésima

Obsérvese que al avanzar la búsqueda la penalización se va haciendo cada vez más rígida.

Las funciones de penalización muy severas no son recomendables en problemas fuertemente restringidos, esto es, en problemas en que hay muchas probabilidades de generar individuos no factibles. Esto se debe a que no sólo se corre el riesgo ya comentado de que el AG gaste la mayor parte de su tiempo en procesar individuos no factibles, sino que además es muy probable que cuando se encuentre un individuo factible éste destaque ostensiblemente sobre los demás por el mero hecho de ser el único factible y provoque una convergencia prematura. Precisamente, la idea de las funciones de penalización se basa en la posibilidad de que el camino hacia buenos individuos factibles pase por la producción de individuos no factibles como estructuras intermedias; si se imponen fuertes penalizaciones la población será rápidamente tomada por individuos factibles y el AG no tendrá tiempo de explotar tales estructuras intermedias.

Por otra parte, las funciones de penalización muy flexibles tienen el riesgo de dejar prosperar a individuos que aun siendo no factibles tengan una aptitud neta mayor que otros factibles.

[1] Se admite por defecto que se está maximizando, para las minimizaciones sería negativa

Existe una técnica llamada ***de filtrado*** que consiste en desechar del proceso a los individuos no factibles que vayan apareciendo. Ésto no es sino un caso extremo de penalización rígida, y comparte (en grado máximo) todos sus inconvenientes.

Algunos autores prefieren usar unas técnicas complementarias a las de penalización consistentes en premiar a los individuos factibles en vez de penalizar a los no factibles. Se las conoce evidentemente como ***técnicas de gratificación*** y su desarrollo es completamente análogo al recién expuesto.

Técnicas de reparación o corrección: Consisten en habilitar un procedimiento con el que corregir cualquier solución no factible que se genere.
Los inconvenientes de estas técnicas residen en que suele ser difícil encontrar un procedimiento de corrección lo suficientemente general y, si se encuentra, normalmente consume muchos recursos de computación. Nótese además que tal procedimiento es necesariamente específico para cada problema.

Técnicas de decodificación: Consisten en realizar la codificación de modo tal que resulte muy poco probable (idealmente imposible) generar soluciones no factibles; dicho de otro modo, se trata de modificar X a través de la codificación para que se aproxime lo máximo posible a $\mathcal{X}$. Normalmente ello conlleva modificar los operadores genéticos para que conserven la factibilidad de los individuos.
Los inconvenientes de estas técnicas son básicamente los mismos que los de las técnicas de reparación.

Obsérvese que de modo indirecto todas estas técnicas están añadiendo conocimiento específico, como ya se anticipó.

Paradigma del recuerdo del comportamiento: Bajo este nombre se conoce a una técnica de consideración de las restricciones específicamente desarrollada para los AGs que consiste básicamente en lo siguiente: se encadenan dos AGs, en el primero la función de aptitud mide exclusivamente el grado de satisfacción de las restricciones de modo parecido a como lo haría una función de gratificación; el segundo AG toma como población inicial la población final del primero —la cual constituye un *recuerdo* de las restricciones— y evoluciona según la aptitud neta del problema, que se ha modificado para que devuelva cero cuando no se satisfagan las restricciones.

3.2.2.1 Ilustración de las técnicas clásicas de tratamiento de los puntos no factibles

Con el fin de concretar ideas, se aplicarán las tres técnicas clásicas de tratamiento de los puntos no factibles al problema binario de la mochila. Como ya quedó dicho, el espacio de búsqueda de este problema queda definido por una restricción, la cual es imprescindible para que el problema tenga contenido. En realidad esta restricción es tanto más imprescindible cuanto menor sea la capacidad C de la mochila. Nótese que con baja capacidad hay mayor probabilidad de que un candidato a solución sea no factible, por lo que el problema estará *fuertemente restringido*.

Este problema se resuelve tradicionalmente mediante técnicas heurísticas. La más sencilla de ellas es el *algoritmo codicioso* o 'glotón', que consiste en introducir los objetos en la mochila por orden decreciente de valor específico ρ_i, comenzando por el objeto de mayor valor específico, hasta que se llene la mochila. Este procedimiento proporciona óptimos aceptables cuando la distribución de valores específicos es uniformemente variada, pero no cuando es dispersa —¿Qué hacer cuando se tienen algunos objetos grandes y muy valiosos?—. Además el rendimiento empeora según crece el tamaño m del problema.

Cuando se resuelve mediante algoritmos genéticos la elección más sencilla para la función de aptitud $u(\mathbf{x})$ consiste en hacerla coincidir con el objetivo $f(\mathbf{x})$, esto es

$$u(\mathbf{x}) := f(\mathbf{x}) = \mathbf{p}' \cdot \mathbf{x} = p_1\, x_1 + \ldots + p_m\, x_m$$

Para considerar la restricción de capacidad limitada $\mathbf{w}' \cdot \mathbf{x} \leq C$ se pueden usar las tres técnicas vistas anteriormente.

Funciones de penalización: Se modifica la función de aptitud así:

$$u(x) \leftarrow u(\mathbf{x}) - Penalty(h(\mathbf{x}))$$

donde $h(\mathbf{x})$ es el grado de violación de la restricción, definido como

$$h(\mathbf{x}) = \begin{cases} \mathbf{w}' \cdot \mathbf{x} - C & \text{sii} \quad \mathbf{w}' \cdot \mathbf{x} > C \\ 0 & \text{sii} \quad \mathbf{w}' \cdot \mathbf{x} \leq C \end{cases}$$

Para el presente problema se han utilizado las siguientes funciones de penalización:

$$\begin{aligned} Penalty_{log}(\mathbf{x}) &:= \log_2(1 + \rho_{max}\, h(\mathbf{x})) \\ Penalty_{lin}(\mathbf{x}) &:= \rho_{max}\, h(\mathbf{x}) \\ Penalty_{sq}(\mathbf{x}) &:= (\rho_{max}\, h(\mathbf{x}))^2 \end{aligned}$$

siendo $\rho_{max} := \max_{i=1,\ldots,m}\{p_i/w_i\}$.

Los resultados experimentales [MICH94,p83] indican que para pequeñas capacidades de la mochila (problema fuertemente restringido) no sirve ninguna de las técnicas de penalización, como ya se comentó. Sin embargo, para problemas en que la capacidad de la mochila es suficiente (arbitrariamente se fija esta 'capacidad suficiente' cuando la solución óptima contiene alrededor de $m/2$ objetos) la función de penalización logarítmica proporciona excelentes resultados para una amplia gama de casos.

Algoritmos de reparación: Dada una solución no factible $\mathbf{x}_{nf}$ se transforma en otra factible $\mathbf{x}_f$ mediante algún procedimiento heurístico de reparación. En el presente contexto, la reparación más sencilla consiste simplemente en extraer objetos de la mochila cuando esté demasiado llena (reparación por extracción). Se han considerado dos estrategias de extracción:

(a) *Reparación aleatoria:* Se construye $\mathbf{x}_f$ extrayendo elementos de la mochila al azar hasta que se vuelva a verificar la restricción de volumen $\mathbf{w}' \cdot \mathbf{x}_f \leq C$.

(b) *Reparación codiciosa:* Se extraen los objetos en orden creciente de valor específico, comenzando por el menos valioso y terminando cuando se vuelva a verificar la restricción de volumen.

Los resultados experimentales indican que el algoritmo codicioso de reparación por extracción es el más apropiado para problemas restringidos. Su eficacia es aceptable con problemas fuertemente restringidos, pero no su rendimiento (eficiencia).

Algoritmos de decodificación: Se trata de cambiar la codificación de los individuos de manera que cada uno pueda ser interpretado como una 'estrategia de incorporación de objetos en la solución', de tal manera que sea muy difícil (idealmente imposible) construir una solución no factible. Para ello se ordenan primero todos los objetos en una lista dinámica de referencia según cierto criterio de prioridad; por ejemplo, si son seis objetos *1, 2, 3, 4, 5* y *6* se pueden ordenar así L ={*4,1,2,5,3,6*}. Después se construye cada individuo como un vector de m componentes en el que la i-ésima componente es un puntero con el que se referencia a la lista dinámica. Siguiendo con el ejemplo, el individuo $\langle \mathtt{4}, \mathtt{3}, \mathtt{4}, \mathtt{1}, \mathtt{1}, \mathtt{1} \rangle$ se decodifica como la siguiente colección de objetos: *5, 2, 6 ,4, 1, 3*. Nótese que la i-ésima componente simpre es un entero positivo no mayor que $m + 1 - i$.
De modo genérico, a este tipo de procedimientos se les llama *codificaciones ordinales* y se tratarán con más detalle en 5.2.2.

La ventaja de esta codificación ordinal estriba en que *los operadores genéticos conservan la factibilidad* de los individuos. Por ello, si se construye una población inicial de puntos factibles (lo cual no es difícil, llevando la cuenta del volumen ocupado) el AG trabajará en todo momento con puntos factibles, y así será el resultado final.

Tan sólo hay que tener en cuenta que, dada la codificación, los operadores genéticos se aplican en el nivel de los *genes*, no de los bits. Así pues, la mutación del i-ésimo gen sólo puede proporcionar un entero del rango [1..m+1-i].

La elección del criterio de prioridades con el que construir la lista dinámica de referencia L influye decisivamente en la rapidez de la búsqueda; a falta de mejores indicaciones se le da un orden aleatorio. Para el presente problema se sabe que es preferible crear una lista L de objetos en orden decreciente de sus valores específicos; a ésto se le dice *decodificación codiciosa ordinal*.

Para asignar a cada individuo su aptitud, esto es, el valor total de todos los objetos que representa, se usa el siguiente procedimiento:

```
BeginProcedure{Eval_x}
 SumP = 0;
 SumW = 0;
 for( i = 1; i <= n; i++) {
  j = x[i];
  RemoveFromTheList(j);
  if( SumW + W[j] <= C ) {
   SumW += W[j];
   SumP += P[j];
  }
 }
EndProcedure
```

Los resultados experimentales son bastante parecidos a los obtenidos con los algoritmos de reparación. Con problemas fuertemente restringidos resulta difícil determinar una población inicial válida, por lo que se debe usar un valor modesto de $PopSize$ y compensarlo con un número mayor de iteraciones. Esta recomendación es extensible para todos los problemas de ese tipo.

Nótese que de modo subrepticio se ha violado el requisito del alfabeto mínimo, ya que el AG está operando con cadenas de *enteros*, no con cadenas binarias. Esto reduce la variedad de esquemas posibles y decrementa consecuentemente el grado de paralelismo implícito.
Además ésta representación es mucho menos natural que la anterior, por lo que se pierden gran parte de las ventajas de análisis asociadas a la naturalidad. La contrapartida se basa en que se ha logrado hacer coincidir el espacio de búsqueda con el dominio del problema, evitando la consideración directa de la no factibilidad.

Existen tres maneras de mejorar estas técnicas clásicas de consideración de los puntos no factibles:

1. Usando algoritmos de penalización, reparación o codificación más sofisticados.

2. Basándose en otras heurísticas mejores que la codiciosa.

3. Combinando *ad hoc* varias de estas técnicas.

A la larga esto obliga a añadir más conocimiento específico, con todos los inconvenientes que eso plantea.

Como puede verse, se vuelve a los mismos inconvenientes de siempre: la especificidad (hay que diseñar un método nuevo para cada tipo de problemas y para cada tipo de restricciones) y la dificultad para tratar eficientemente problemas fuertemente restringidos. Este es uno de los motivos que impulsan a extender el paradigma de los algoritmos genéticos, como se verá en el capítulo 5.

3.2.3 Elección de la población inicial

En términos generales, la población inicial debe ser lo más variada posible. Es necesario que la distribución de aptitudes sea uniforme para evitar la convergencia prematura, y también es necesario que haya variedad de esquemas. En la práctica, a falta de más información se elige la población inicial al azar. Una vez más, el conocimiento específico puede ayudar para elegir una población inicial factible y/o próxima al óptimo.

Conviene que la población inicial contenga la mayor cantidad posible de puntos factibles. En ocasiones —vg., en problemas fuertemente restringidos— es difícil obtener puntos factibles y se deberá comenzar a iterar con una cantidad muy reducida de ellos. En caso de que eso ocurra no es conveniente usar un tamaño de población muy pequeño, dado que perjudica gravemente al requisito de diversidad, ni tampoco introducir una gran cantidad de puntos no factibles, ya que se reduce grandemente la eficiencia del AG. Tradicionalmente se ha resuelto este problema repitiendo los puntos factibles en la población inicial; ello no elimina los inconvenientes señalados anteriormente, pero reduce su peligrosidad. Recientemente se ha desarrollado una variedad de algoritmos genéticos que hace evolucionar el tamaño de la población, permitiendo comenzar con una población inicial pequeña e irla ampliando según progrese la evolución. Se estudiarán en la sección 4.1.

Aún queda una cuestión por resolver: Para una población inicial dada $P[0]$ ¿Está garantizada la convergencia del AG al óptimo? O mejor aún, ¿Es posible modificar el AG de modo que converja siempre, independientemente de la población inicial? La respuesta a ésta cuestión es que sí es posible conseguir convergencia independientemente de la población inicial si se le añade un mecanismo llamado ***prueba de contractividad***: Planteando el AG como una iteración de cierta transformación de una población en otra —el bucle básico—, se trata de adaptarlo para que cumpla las condiciones del teorema del punto fijo de Banach, con el cual se garantiza la convergencia.

Para centrar la cuestión, el Teorema de Banach afirma que toda aplicación contractiva $f : \mathcal{S} \to \mathcal{S}$ sobre un espacio métrico completo $\langle \mathcal{S}, \delta \rangle$ tiene un único punto fijo $x_f \in \mathcal{S}$ el cual verifica que

$$(\forall x_0 \in \mathcal{S}) \qquad \lim_{k\to\infty} f^{[k]}(x_0) = x_f$$

donde se definen

$$f^{[0]}(x) := x \qquad\qquad f^{[k]}(x) := f\Big(f^{[k-1]}(x)\Big) \qquad (\forall k = 1, 2, \ldots)$$

De este modo, si se encontrase un espacio métrico apropiado en el que el bucle básico del AG fuera contractivo entonces quedaría demostrada la convergencia del algoritmo hacia el mismo punto fijo independientemente de la población inical $P[0]$; además aunque el problema tuviera varios óptimos, quedaría garantizada la convergencia hacia uno cualquiera de ellos.

Evitando los detalles técnicos, es posible demostrar que existen tales espacios métrico en el que el bucle básico del AG es contractivo. Para ello basta con hacer una imposición adicional al bucle consistente en considerar como válida una iteración si y sólo si hay una mejora absoluta en la aptitud media de la población, esto es:

$$AveFitn(P[t]) < AveFitn(P[t+1])$$

donde, como ya se sabe,

$$AveFitn(P[t]) := \frac{1}{PopSize} \sum_{\mathbf{v} \in P} Fitness(\mathbf{v}[t])$$

En caso de que no se produzca la mejora, no se cuenta la iteración, es decir, se ejecuta de nuevo el bucle de selección+recombinación.

A efectos prácticos, el AG con requisito de contractividad para el bucle básico queda así:

```
BeginAlgorithm{ContractiveGeneticAlgorithm}
 P[0] = InitPop();
 FitP[0] = EvalPop(P[0]);
 for( t = 0; t < MaxNumGen; t++) {
  Q[t] = SelectBreedersFrom(P[t]);
  Q[t] = Crossover(Q[t]);
  Q[t] = Mutate(Q[t]);
  FitQ[t] = EvalPop(Q[t]);
  P[t] = SelSurv(P[t], Q[t], FitP[t], FitQ[t]);
  FitP[t] = EvalPop(P[t]);
  if( ChkTermCond(P[t], FitP[t]) )
   return;
  if( FitP[t] <= FitP[t-1])              Prueba de contractividad
   t--;
 }
EndAlgorithm
```

De lo anteriormente expuesto cabría concluir que con la prueba de contractividad la elección de la población inicial $P[0]$ es indiferente, dado que el AG convergerá de todos modos. Esto no es completamente cierto, ya que habrá grandes diferencias en la velocidad de convergencia según sea la población inicial elegida. Desafortunadamente, no hay modo fácil de saber a priori si cierta población inicial proporcionará una mayor o menor velocidad de convergencia. De hecho, puede ocurrir en la práctica que el algoritmo se atasque en cierto valor de t porque los operadores genéticos sean incapaces de mejorar la aptitud media. Comúnmente eso se resuelve deshabilitando temporalmente la prueba de contractividad, y si sigue ocurriendo es señal de que se debe recurrir a otros criterios de contractividad, la mayoría de los cuales son dependientes del problema.

3.2.4 Convergencia en los AGs y criterios de terminación

Hasta el momento se ha aceptado por defecto como criterio de terminación de los AGs el del máximo número de iteraciones. Se admite que, dada la hipótesis de la construcción por bloques y una adecuada población inicial, los mejores individuos de la población se acercarán cada vez más al óptimo. Es decir, se asume implícitamente que al llegar a cierto valor de t el algoritmo ha convergido a efectos prácticos. Ésta manera de determinar la convergencia es excesivamente arbitraria en muchos casos, lo que lleva a plantear otros criterios de terminación.

Tradicionalmente se distinguen dos tipos de criterios de terminación:

- ***Criterios de terminación enfocados al coste:*** Abreviadamente, criterios de 'tipo MAX'. Son los que limitan a priori el número máximo de iteraciones ($MaxIter$) o el máximo número de evaluaciones ($MaxNrEval$, habitualmente usados cuando la función de evaluación es complicada). Son los más sencillos, pero no siempre tienen significado, ya que normalmente no hay ninguna razón para dar a dichos números máximos uno u otro valor. En tal circunstancia o bien ocurrirá que el AG "se quede corto", esto es, no esté al terminar lo suficientemente cerca del óptimo, o bien consumirá recursos computacionales inútilmente estando ya lo suficentemente cerca del óptimo.

- ***Criterios de terminación enfocados a la calidad:*** También se les conoce como *criterios incrementales de terminación* o, abreviadamente, criterios de 'tipo TOL'. Se hace parar al algoritmo una vez que se hayan superado ciertos requisitos de convergencia, independientemente de la iteración en que se encuentre. Dependiendo del criterio práctico que se utilice para evaluar la convergencia se subdividen en dos tipos:

  – *Enfocados a la estructura:* Se evalúa la convegencia de la población verificando la cantidad de genes que han convergido. Se considera que un gen del genotipo ha convergido cuando cierto porcentaje de la población (alrededor del 90%) tiene los alelos iguales en dicho gen (o parecidos, si la representación no es binaria). La búsqueda termina cuando el número de genes que han convergido excede cierto porcentaje (alrededor del 95%) del total de genes de que consta el genotipo.

– *Enfocados al contenido:* Miden el progreso hecho por el algoritmo en cierto número de iteraciones; si es menor que cierta tolerancia TOL, que es un parámentro del método, la búsqueda ha terminado. Habitualmente se mide el 'progreso hecho por el algoritmo' como el incremento relativo de la aptitud total de la población; también se puede usar la aptitud máxima o la mínima, según sea el problema.

La elección de unos valores u otros para las tolerancias es delicada: no sólo depende del problema, sino que también influye la suerte que se haya tenido en la ejecución del AG (recuérdese que son métodos estocásticos). Además, una pequeña reducción de las tolerancias permitidas puede conllevar un tiempo de ejecución mucho más grande.

Habitualmente, se usa un criterio de terminación híbrido que consta de una tolerancia y un mecanismo de seguridad del tipo MAX para cuando se alargue demasiado la búsqueda.

3.2.4.1 Elitismo

El método más utilizado para mejorar la convergencia de los AGs es el ***elitismo***. Consiste básicamente en realizar la etapa de selección en dos partes:

1º. Se muestrea una élite de r miembros de entre los mejores de la población inicial y se incorporan directamente a la población final, sin pasar por la población intermedia.

2º. La población auxiliar de criadores se muestrea de entre los $n - r$ restantes miembros de la población inicial

Comúnmente el tamaño de la élite r es bastante pequeño (1 ó 2 para $n = 50$), y el tipo muestreo es bien directo o bien por sorteo, ambos en la variedad diversa.

La figura 3.2 muestra cómo se modifica el bucle básico del funcionamiento de un AG al introducir ese mecanismo y la figura 3.3 muestra la implantación genérica de un algoritmo de ese tipo.

Bajo ciertas condiciones muy generales, la introducción del elitismo garantiza la convergencia teórica al óptimo global; en la práctica, mejora la velocidad de convergencia de los AGs cuando la función de evaluación es unimodal —esto es, no existen subóptimos—, sin embargo la velocidad de convergencia empeora con funciones fuertemente multimodales.

Con tamaños de población pequeños se consiguen efectos similares a los del elitismo introduciendo ***reinicializaciones*** periódicas en los AGs: cada vez que el AG converge se salvan los mejores individuos, se reinicializan los demás y se

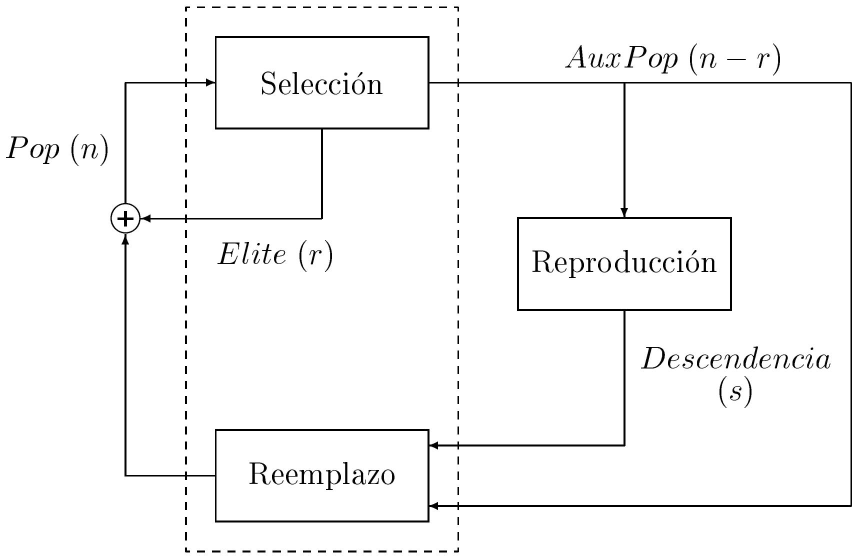

Figura 3.2: Bucle básico de un AG elitista

```
BeginAlgorithm{ElitistGeneticAlgoritm}
 P[0] = InitPop();
 FitP[0] = EvalPop(P[0]);
 for( t = 0; t < MaxNumGen; t++){
  Q[t] = SelectBreedersFrom(P[t]);
  R[t] = SelectEliteFrom(P[t]);
  Q[t] = Crossover(Q[t]);
  Q[t] = Mutate(Q[t]);
  FitQ[t] = EvalPop(Q[t]);
  P[t] = SelSurv(P[t], Q[t], FitP[t], FitQ[t]);
  P[t] = AddElite(P[t],R[t]);
  FitP[t] = EvalPop(P[t]);
  if( ChkTermCond(P[t], FitP[t]) )
   return;
 }
EndAlgorithm
```

Figura 3.3: Estructura general de un AG elitista vuelve a comenzar. La reinicialización tiene efectos beneficiosos sobre las prestaciones del método debido a que introduce diversidad, requisito especialmente crítico en los AGs con poblaciones pequeñas.

3.2.5 Elección de las funciones de evaluación y de aptitud

Como ya se indicó en la sección 2.1 para poder ejecutar un AG es preciso tener definida una función de evaluación de individuos y una función de aptitud que se deriva directamente de ella por desplazamiento y escalado. Es necesario introducir la distinción entre la función de aptitud y la de evaluación por dos motivos:

1º. Para hacer que las evaluaciones tomen valores positivos (ver apdo. 2.3.4).
2º. Para controlar la diversidad de la población.

3.2.5.1 Dificultades en la elección de la función de evaluación

En principio, la función de evaluación se hace coincidir con el objetivo a maximizar; sin embargo, no siempre es fácil concretar numéricamente dicho objetivo. Considérese por ejemplo el *problema de la satisfabilidad de una expresión lógica en forma normal conjuntiva*:
Se dice que una expresión lógica está en forma normal conjuntiva (CNF) cuando se expresa como una sucesión de cláusulas separadas por el operador lógico $\wedge$ (AND). Una cláusula es una sucesión de términos separada por el operador lógico $\vee$ (OR), un término es una variable lógica o su negación y una variable lógica es aquélla que sólo puede tomar dos valores T (verdadero) ó F (falso).
Pues bien, el problema consiste en determinar, para una expresión lógica CNF una asignación de variables de manera que el valor total de la expresión sea T. Por ejemplo, para la expresión lógica

$$Expr(a, b, c, d, e) := (a \wedge \neg b \wedge c) \vee (b \wedge c \wedge d \wedge \neg e) \vee (\neg a \wedge c) \vee (a \wedge c \wedge \neg e)$$

cualquier asignación de la forma $(a, b, c, d, e) \leftarrow (T, *, T, *, *)$ proporciona un valor verdadero (T).

Si se aplicara un AG para resolver un problema de este tipo, es evidente qué representación habría que elegir: un vector binario con tantos elementos como variables distintas tenga la expresión. Como no hay restricciones cualquier individuo será factible, lo que evita tener que recurrir a métodos de reparación o decodificación. Sin embargo, no es en absoluto evidente qué función de evaluación se debe elegir. No sirve la propia expresión lógica, ya que presumiblemente todas las posibles soluciones toman el valor F —si alguna tomara T el problema ya estaría resuelto— y no parece haber una manera sencilla de establecer un 'criterio de calidad' para los candidatos a solución, pues todos parecen igual de malos. En definitiva, el problema de satisfabilidad CNF así planteado no tiene admite una función de evaluación natural. Se han desarrollado diversas maneras de replantear este problema y otros similares con el fin de que se les pueda asignar una función de evaluación. En [PROC89,p125] se dan indiciaciones para llevar a cabo ese replanteamiento.

No acaban aquí los problemas con la función de evaluación. Como fácilmente se puede comprender, dicha función debe ser fácil de manejar en la práctica. Ocurre en ocasiones que hay un objetivo claramente definido, pero la determinación explícita de su valor para cada individuo es prolija. En tal circunstancia no se puede tomar ese objetivo como función de evaluación, porque sería inoperante. Afortunadamente, los AGs son lo suficientemente insensibles al error en la aptitud como para poder determinar la función de evaluación aproximando (vg. por linealización) la función objetivo sin menoscabo de las prestaciones.
Un caso muy común en que ésto ocurre es en el *problema generalizado del control óptimo*. En él se debe maximizar cierto índice de utilidad $J(\mathbf{x}, \mathbf{u})$ en el que $\mathbf{x}$ es un vector de variables de estado y $\mathbf{u}$ es un vector de variables de control (o decisión) ligadas a través de las ecuaciones del modelo $\mathbf{g}(\mathbf{x}, \mathbf{u}) = \mathbf{0}$. Para calcular la utilidad del sistema en cierto estado $\mathbf{x}$ es necesario resolver las ecuaciones del modelo, lo que suele requerir largos cálculos. Para evitar estos cálculos se linealiza en torno a un estado de equilibrio y se trabaja con el modelo incremental resultante:

$$\Delta\mathbf{x} = -\left(\frac{\partial\mathbf{g}}{\partial\mathbf{x}}\right)^{-1}\left(\frac{\partial\mathbf{g}}{\partial\mathbf{u}}\right)\cdot\Delta\mathbf{u}$$

Si la función objetivo es también complicada de evaluar, cabe aproximarla de la misma manera:

$$\Delta J = \frac{\partial J}{\partial\mathbf{x}}\Delta\mathbf{x} + \frac{\partial J}{\partial\mathbf{u}}\Delta\mathbf{u}$$

Para aproximar la aptitud de un descendiente se debe combinar de alguna manera (vg., promediando) la información del modelo de cada uno de los progenitores.

Por último, dado que los AGs maximizan aptitudes, se admite por defecto que se maximiza el objetivo. En ocasiones resulta más apropiado plantear el problema como uno de *minimización* de cierto objetivo —en general, siempre que sea equiparable a un coste—. Es inmediato transformar estos problemas en otros de maximización sin más que cambiarle el signo a la función de costes. A veces se usa también como función de evaluación el inverso de la función de costes; es interesante esta opción porque proporciona una referencia visible común para todas las aptitudes (el cero), pero exige que la función de costes sea estrictamente positiva.

3.2.5.2 Desplazamiento y escalado de la función de evaluación

Una vez elegida la función de evaluación, se construye la función de aptitud desplazándola y escalándola convenientemente.

El ***desplazamiento*** tiene como finalidad principal hacer que la función de aptitud devuelva valores positivos, tal como exigen las hipótesis del Teorema Fundamental de los Algoritmos Genéticos. Se puede demostrar que, bajo condiciones muy generales, si se da una aptitud nula a las evaluaciones negativas el Teorema Fundamental se sigue verificando, pero en la práctica este remedio se usa raras veces porque frena la convergencia del AG. El procedimiento más usado para hacer positivas las evaluaciones es el de la *ventana de desplazamiento*. Consiste en desplazar las evaluaciones así:

$$Fitness(\mathbf{v}) = F - Eval(\mathbf{v}) \qquad (\forall \mathbf{v} \in P)$$

donde F es una cota superior para las evaluaciones, esto es,

$$(\forall t) \quad \forall \mathbf{v} \in P \qquad Fitness(\mathbf{v}[t]) \leq F$$

No es habitual conocer a priori el rango de valores que va a tomar la función de evaluación, por lo que es preferible ir adaptando dinámicamente el valor de F del siguiente modo:

$$Fitness(\mathbf{v}[t]) = F_{max}[t] - Eval(\mathbf{v}[t]) \qquad (\forall \mathbf{v}[t] \in P[t])$$

donde $F_{max}[t]$ es la máxima aptitud de cualquier individuo evaluado hasta la iteración inmediatamente anterior, $t-1$.

Ahora bien, si el valor de $F_{max}[t]$, o en su caso el de F, no está lo suficientemente cerca de la máxima aptitud de la presente generación, la búsqueda se frena e incluso se puede estancar. La causa está una vez más en la pérdida de la diversidad de aptitudes: Efectivamente, si $F_{max}[t]$ es mucho mayor que cualquier evaluación presente, las correspondientes aptitudes serán todas muy parecidas, se reducirá la presión selectiva hacia los mejores individuos y la búsqueda se estancará, con el consiguiente riesgo de deriva genética. Supóngase, por ejemplo, que $F_{max}[t] = 100$ pero que en el presente instante la población sólo contiene individuos cuyas evaluaciones proporcionan valores entre 5 y 10. Evidentemente las aptitudes variarán entre 90 y 95. Sin embargo, si fuera $F_{max}[t] = 15$ las aptitudes estarían comprendidas entre 5 y 10, y el individuo más apto tendría ahora doble aptitud que el menos apto. En resumen, se ha mejorado la capacidad de discriminación de aptitudes.

En definitiva, resulta conveniente ir actualizando el valor de F_{max} cada cierto número de generaciones con el fin de evitar una posible pérdida de la diversidad. El periodo de actualización W es un parámetro del método que suele oscilar entre 0 y 5, conviniéndose en que $W = 0$ representa al caso en que F_{max} nunca se actualiza.

Como ya se comentó, una de las maneras de controlar la diversidad de las aptitudes es definiendo una función de aptitud neta, distinta de la función de evaluación. Esto es lo que se conoce como ***escalado*** de la función de evaluación. Mediante el escalado se trata de evitar un fenómeno con causas opuestas al anterior pero de similares efectos: es el fenómeno ya descrito de la convergencia prematura hacia superindividuos. No obstante, la finalidad del escalado es más general; siguiendo a Goldberg [GOLD89,p122],

> Desde el estudio pionero de De Jong, el escalado de los valores de la función objetivo se ha convertido en una práctica ampliamente aceptada. Se hace ésto para mantener los niveles apropiados de competición a lo largo de una simulación. Sin escalado, en las fases tempranas hay una tendencia a que unos pocos superindividuos dominen el proceso de selección. En este caso, los valores de la función objetivo deben ser subescalados para evitar que tales superindividuos ocupen la población. Más tarde, cuando la población ha convergido casi en su totalidad, la competición entre los miembros de la población es menos intensa y la simulación tiende a evolucionar erráticamente. En este caso los valores de la función objetivo deben ser sobreescalados para acentuar las diferencias entre los miembros de la población con el fin de continuar favoreciendo a los mejores miembros.

En último término se trata de controlar los efectos de la presión selectiva, amortiguándolos en las fases tempranas de la evolución y acentuándolos en las fases tardías.

Es común incluir el desplazamiento, directa o indirectamente, en el propio algoritmo de escalado. En caso de que no se haga así, la fase de desplazamiento es previa a la de escalado.

Mecanismos de escalado: Esencialmente se usan cuatro mecanismos de escalado. Por orden de popularidad son:

Escalado lineal: Se calcula la aptitud a partir de los valores proporcionados por la función de evaluación del siguiente modo

$$Fitness(\mathbf{v}) := a \cdot Eval(\mathbf{v}) + b$$

donde los parámetros a y b se calculan para cada generación de modo tal que se verifiquen estas dos condiciones

1º. Deben coincidir los valores medios de la función de evaluación y de la función de aptitud en la población. Desarrollando ambos valores es inmediato comprobar que necesariamente

$$b \equiv AveEval(P) \cdot (1 - a)$$

Así se garantiza que un elemento típico de la población contribuirá en promedio con un descendiente en la próxima generación.

2º. Para controlar la presión selectiva se obliga a que

$$MaxFitn(P) \equiv C \cdot AveFitn(P)$$

donde el parámetro de control C toma valores entre 1.2 y 2.0 para tamaños de población típicos (ie., entre 50 y 100 miembros). Eso hace que el miembro más apto de la población obtenga en promedio C descendientes en la próxima generación.
También es fácil comprobar que esa condición obliga a hacer

$$a \equiv (C-1)\frac{AveEval}{MaxEval - AveEval}$$

Con el escalado lineal se consigue subescalar las evaluaciones en las fases tempranas de la evolución y sobreescalarlas en las fases tardías. Desafortunadamente, es común que el sobreescalado lineal proporcione aptitudes negativas para los peores miembros. Cuando esto ocurra, se obtienen resultados similares sustituyendo la segunda condición por esta otra:

2º. Se hace que la aptitud del peor miembro valga cero, ie.,

$$MinFitn(P) \equiv 0$$

lo que obliga a hacer

$$a \equiv \frac{AveEval}{AveEval - MinEval}$$

Truncación sigma: Para evitar el problema de las evaluaciones negativas en las fases tardías de la evolución se ha sugerido utilizar la desviación estándar (σ) de las evaluaciones en el cálculo de la aptitud así:

$$Fitness(\mathbf{v}) := Eval(\mathbf{v}) - AveEval(P) + c \cdot StdDevEval(P)$$

En esa fórmula la constante c se elige como un múltiplo razonable de la desviación estándar (entre 1 y 3) y a los resultados negativos se les da arbitrariamente el valor cero. De modo análogo al escalado lineal, la truncación sigma consigue evitar el dominio prematuro de superindividuos mientras que en fases posteriores favorece una sana rivalidad entre individuos similares.

Escalado potencial: Se calcula la aptitud como una potencia de la función de evaluación

$$Fitness(\mathbf{v}) := (Eval(\mathbf{v}))^k$$

El parámetro k es próximo a uno (vg. $k = 1.005$), si bien su valor concreto depende en gran medida del problema en particular.

Métodos de escalado basados en el orden: Son un conjunto de métodos de escalado que calculan la aptitud de cada individuo como una función sencilla de la posición ($Rank(\mathbf{v})$) que ocupa cada uno en una lista ordenada. Los criterios de ordenación son diversos, pero el más usado con diferencia es [PROC89,p119] por orden decreciente de aptitudes brutas, ie.,

$$Fitness(\mathbf{v}) := g\left(Rank(\mathbf{v})\right)$$

Al mejor individuo se le designa como $\mathbf{v}_{\#1}$ y al peor $\mathbf{v}_{\#PopSize}$.

La función $g(\cdot)$ suele ser lineal, de este modo la distribución de aptitudes será uniforme:

$$Fitness(\mathbf{v}_{\#k}) := MaxFitn - c \cdot (k-1) \qquad (\forall k = 1, \ldots, PopSize)$$

Los parámetros $MaxFitn$ y c permiten controlar la presión selectiva a través de la diversidad inducida: cuanto mayor sea c y/o menor sea $MaxFitn$ mayor será la presión selectiva.

En ocasiones es preferible utilizar una distribución geométrica de aptitudes, es decir:

$$Fitness(\mathbf{v}_{\#k}) := MaxFitn \cdot c^{k-1} \qquad (\forall k = 1, \ldots, PopSize)$$

donde c es un parámetro positivo no mayor que 1 con el que se controla la presión selectiva; los valores típicos están alrededor de 0.90.

Otra distribución alternativa es:

$$Fitness(\mathbf{v}_{\#k}) := MaxFitn \cdot (1 - MaxFitn)^{k-1} \qquad (\forall k = 1, \ldots, PopSize)$$

con $MaxFitn \in (0,1)$. Esta alternativa se usa mucho con tamaños grandes de población, ya que se evita la normalización de las aptitudes debido a que aproximadamente

$$\sum_{k=1}^{PopSize} MaxFitn \cdot (1 - MaxFitn)^{k-1} \approx 1$$

Como era de esperar, cuanto mayor sea $MaxFitn$ más fuerte será la presión selectiva.

Para fijar ideas, los métodos de escalado basados en el orden presentan las siguientes ventajas:

- Proporcionan un método sencillo y natural de controlar la diversidad de la población.
- Como consecuencia de lo dicho anteriormente se puede ejercer un mejor control sobre la presión selectiva.

- También se puede controlar con facilidad la amplitud de la búsqueda.

y también los siguientes inconvenientes:

- Como lo único que importa es el orden relativo de los individuos, se ignora sistemáticamente la información sobre las respectivas diferencias de aptitudes, lo que suele ralentizar la búsqueda.

- Recae sobre el usuario la decisión de cuándo usarlos, y de cuál de ellos usar, llegado el caso.

3.2.6 Mejoras a los operadores genéticos

En el apartado 2.3.3 se dio una justificación teórica de los operadores genéticos. Allí se comentaba que el cruce se introduce para proporcionar un mecanismo de intercambio estructurado de información útil (bloques constructivos) entre individuos; por su parte, la mutación permite explorar nuevas zonas del espacio de búsqueda a las que posiblemente nunca se llegaría usando sólo el cruce, y además proporciona un mecanismo de seguridad frente a pérdidas de información valiosa.

Se van a estudiar en este apartado posibles mejoras a la búsqueda genética clásica con el fin de hacer más eficientes los mecanismos de procesamiento de esquemas, desde cuya perspectiva los operadores genéticos clásicos adolecen de al menos dos defectos:

1. El cruce clásico no puede realizar ciertas combinaciones de bloques constructivos. Supóngase por ejemplo que los esquemas `001*****01` y `****11****` tienen alta aptitud; es evidente que por medio del cruce clásico resulta imposible obtener el esquema que los combina `001*11***01`.

2. Existe una asimetría entre la aplicación del cruce y la mutación: la unidad básica de mutación es el gen, la del cruce es el individuo; en consecuencia, la probabilidad de que un individuo se cruce no depende de su longitud, mientras que la probabilidad de que contenga genes mutados es más alta cuanto más largo sea.

3.2.6.1 Alternativas al cruce clásico

Se han propuesto muchas alternativas que mejoran, en circunstancias más o menos generales, las prestaciones del cruce clásico. Las alternativas más importantes son:

Cruce multipunto: Consiste en cruzar los individuos en torno a dos o más puntos. Este cruce mejora la capacidad de procesado de esquemas, pero a costa de perder velocidad de convergencia.

Cruce segmentado: Es una versión del cruce multipunto que permite la introducción de variabilidad en el número de puntos de cruce. Consiste en reemplazar el número fijo de puntos de cruce por una *probabilidad de segmentación* p_{seg} que da cuenta de la posibilidad de que se produzca un cruce al llegar a cierto punto de la cadena. Así se trata de corregir la asimetría con respecto a la mutación.
Por ejemplo, si la probabilidad de segmentación es $p_{seg} = 0.20$ entonces el promedio esperado de cruces será $0.2Len$ en cada par de progenitores, aunque no necesariamente tomará ese valor.

Cruce uniforme: Para cada bit del primer descendiente se decide, con probabilidad p_{unif}, de qué progenitor heredará su valor en esa posición. El segundo descendiente recibe el correspondiente gen del otro progenitor.
Dado que el cruce uniforme intercambia más bien genes que bloques constructivos, se usa para combinar atributos específicos, con independencia de la posición en que han sido codificados. En algunos problemas esta capacidad compensa el riesgo adicional que se introduce de descomponer bloques constructivos. No obstante se recomienda utilizar el cruce uniforme sólo en casos concretos en los que haya motivos fundados para hacerlo.

A continuación se presentan cinco mecanismos adicionales que se pueden añadir a cualquiera de los operadores de cruce:

Cruces anulares: En muchas ocasiones no hay ninguna razón para considerar que el primer y el último bit de la cadena no son adyacentes, de manera que resulta más natural utilizar una *representación anular*, esto es, con forma de anillo. Ocurre entonces que el cruce clásico introduce un sesgo porque obliga a intercambiar segmentos a partir de una posición fija —la que marcan los extremos—. Para evitarlo se deben cruzar los individuos en un número par de posiciones. En ésto consiste el cruce en anillo.

Cruces externos: Independientemente de la posición y la extensión —número de bits— que ocupe cada gen en la estructura de los individuos, el cruce en los AGs se realiza en el nivel de bits, no en el de genes. Se hace así para trabajar con la mayor cantidad posible de esquemas. Sin embargo, en algunos problemas (vg., el de la mochila del apartado 3.2.2.1) es más natural obligar a que los cruces se produzcan exclusivamente en los puntos de separación entre genes; precisamente en ésto consisten los *cruces externos*. Eso es equivalente a incrementar el alfabeto de símbolos, lo que conlleva una notable reducción de la propiedad de paralelismo implícito. Por contra, el uso de una representación más natural hace que se procesen más esquemas útiles, hasta el punto de que se puede compensar la pérdida de capacidad de procesamiento en paralelo. En el capítulo 5 se estudiará esta cuestión con detenimiento.

Cruces con barajado: Antes de realizar el cruce se desordenan aleatoriamente las posiciones de ambos progenitores a la par. Después se realiza el cruce y finalmente se deshace el barajado.

Cruces con adaptación: Se sustituyen las estrategias de asignación aleatoria de los puntos de cruce por estrategias adaptativas de asignación análogas a las del propio AG. Se hace ésto con la esperanza de que los puntos de cruce experimenten un proceso de evolución: si cierto punto ocasiona descendencia de baja calidad, se extingue; en caso contrario prospera.

Cruces solitarios: Se trata simplemente de prohibir la aplicación de mutaciones a individuos procedentes de cruces. Esto introduce orden en el proceso de búsqueda y elimina la asimetría de aplicación entre ambos operadores. Naturalmente, nada impide definir un nuevo operador *cruce+mutación* que ejecute ambas operaciones con cierta probabilidad.
Es común utilizar este enfoque cuando el AG contiene otros operadores genéticos dependientes del problema, aunque sólo sea por sencillez y orden en la implantación. Esto no es sino otra forma del modelo de *reproducción operacional*, ya comentado en 2.1.3.

3.2.6.2 Alternativas a la mutación clásica

La mutación es un operador de importancia teórica secundaria, lo que se manifiesta en su baja probabilidad de aplicación en comparación con el cruce. Ello no quita para que en la práctica sea un operador imprescindible; de hecho, se ha comprobado que un AG puede funcionar sin realizar cruces ($p_{xov} = 0$), pero no sin realizar mutaciones.

Mutaciones sobre genes: Son análogas al cruce externo en la medida en que afectan a un gen completo, no a un bit. Dado que un gen puede tomar más de dos valores (alelos), la mutación se realiza sustituyendo el valor actual por otro más o menos parecido. El procedimiento concreto con que se introduce dicha alteración depende de la codificación utilizada; el más sencillo consiste en añadir o sustraer, con igual probabilidad, una cantidad fija al entero con que se codifica el correspondiente alelo. Ello presupone que alelos parecidos se codifican mediante enteros contiguos.

Mutaciones no estacionarias: Es un hecho admitido que la importancia de la mutación es menor en las fases tardías de la búsqueda que en las tempranas. Esto se explica porque es en las primeras etapas cuando en AG debe realizar una búsqueda lo más amplia posible mientras que en las últimas la búsqueda debería

ser más local. Por eso en ocasiones conviene ir reduciendo la tasa de mutación p_{mut} según progresa el AG.
La manera en que se lleva a cabo la reducción depende en gran medida del problema; la más sencilla consiste en multiplicar la tasa de mutación por un factor de reducción constante cada cierto número de generaciones.

Mutaciones no uniformes: Consisten en dar distintas probabilidades de mutación a cada gen —o a cada bit dentro de un gen—, dependiendo de su significado. En [PROC89,p104] se estudian varios modos de asignar esas probabilidades.

3.2.7 Mejoras a los mecanismos de selección

En un AG se distinguen tres mecanismos de selección: selección de élite, selección de criadores y selección de supervivientes (reemplazo). Por defecto la selección de la élite se lleva a cabo por muestreo directo, la de los criadores por sorteo y la de los supervivientes por reemplazo inmediato.

Interesa especialmente tener un buen mecanismo de selección de criadores, pues ello hará posible un control apropiado de la diversidad de aptitudes de la población. Concretamente, conviene que el mecanismo de selección sea lo más insensible posible a la distribución de aptitudes y a los errores de muestreo. Como ya se vio en 3.1.2.2 la insensibilidad a la distribución de aptitudes se obtiene modificando el mecanismo con el que se asignan dichas aptitudes; análogamente, si se actúa directamente sobre los mecanismos de muestreo, se incrementa la insensibilidad ante los correspondientes errores. Nótese que ambos mecanismos —de muestreo y de asignación de aptitudes— están relacionados, de manera que al controlar uno de ellos se mitigan en parte los efectos del otro.

3.2.7.1 Mejoras al muestreo por sorteo. Selección con contadores

El muestreo de criadores por sorteo se utiliza para obtener un crecimiento exponencial de los esquemas aventajados. Ocurre en la práctica que, debido a la finitud de la población, el crecimiento de los esquemas aventajados es apreciablemente menor; además, el muestreo por sorteo no es insensible a la distribución de aptitudes y a los errores derivados de la finitud de la población.

Se han propuesto otras técnicas de muestreo con un rendimiento en poblaciones finitas similar a la de por sorteo y que resultan ser más insensibles que ésta; las más apropiadas parecen ser la técnica de muestreo universal, la de muestreo por restos y en especial el muestreo por valor esperado. Las dos primeras ya fueron descritas a su debido tiempo en 2.1.2, en cuanto al ***muestreo por valor esperado*** es una variante del muestreo por sorteo adaptada especialmente para la fase de selección de criadores de los algoritmos genéticos. Consiste en asignar a cada uno de los individuos unos contadores que se inicializan a $\lceil 2 \cdot p_i \rceil$. Cada vez que un individuo de la población es elegido para reproducirse se decrementa el contador en 1 ó en 2, según el operador de reproducción empleado (habitualmente 1 para el cruce y 2 para la mutación); cuando el contador de un individuo llegue a cero, se le excluye de posteriores muestreos. Este método de muestreo porporciona un criterio de selección de excelentes prestaciones en la práctica, por lo que es ampliamente utilizado. Por motivos obvios a dicha selección se le llama a veces *selección con contadores.*

3.2.7.2 Selección de estado estacionario

La idea consiste en modificar la estructura genérica del AG seleccionando directamente los progenitores sin necesidad de pasar por una población intermedia, tal y como se muestra en la figura 3.4. La población de progenitores es bastante pequeña (nunca mayor que el 10% de *PopSize*) y el reemplazo se realiza seleccionando por sorteo inverso (ie., las puntuaciones son inversamente proporcionales a las respectivas aptitudes) $OffsprSize$ elementos de entre los *PopSize* de la población (ie., s de entre n), que serán reemplazados por la descendencia.

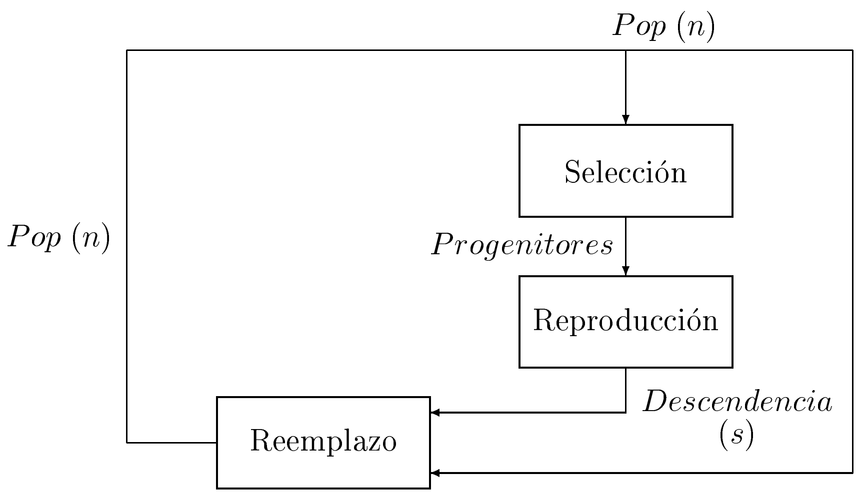

Figura 3.4: Estructura genérica de un AGEE

Ésta metodología constituye más bien una nueva variedad de algoritmos genéticos, los Algoritmos Genéticos de Estado Estacionario (AGEE); se estudia aquí porque afecta directamente a los mecanismos de selección.

Los AGEE presentan las siguientes ventajas sobre los AGs de estructura clásica:

- Son más insensibles a los errores de muestreo, es decir, asignan con precisión probabilidades de reproducción proporcionales a la aptitud.
- El método de reemplazo proporciona un mecanismo de elitismo implícito con el que se protege a los individuos más aptos.
- Tienen menor riesgo de convergencia prematura, lo que permite trabajar presiones selectivas moderadamente altas.
- Se preservan las condiciones del Teorema Fundamental de los Algoritmos Genéticos.

3.2.7.3 Selección con carga genética

La idea consiste en forzar a mantener cierto grado de diversidad haciendo que la población contenga algunos individuos selectivamente neutros (la carga genética). La implantación se realiza de modo parecido al elitismo, esto es, haciendo que algunos miembros de la población inicial pasen directamente a la población auxiliar y realizando la selección de los criadores que quedan libres entre los restantes miembros. Los individuos neutros se generan en la fase de inicialización y se suelen renovar cada cierto tiempo.

En la sección 4.1 se estudiará una forma completamente distinta de realizar la selección de supervivientes con la que se evitan gran parte de los inconvenientes expuestos.

3.2.8 Parametrización de los AGs

Para que un algoritmo genético pueda funcionar se deben dar valores apropiados a un conjunto de parámetros de entre los que destacan el tamaño de la población ($PopSize$), la longitud de los individuos (Len), las probabilidades de mutación y cruce (p_{mut} y p_{xov}), el máximo número de iteraciones ($MaxIter$), la precisión (TOL), el periodo de actualización de la ventana de desplazamiento (W) y otros específicos de cada método como, por ejemplo, los coeficientes de la función de penalización o de la función de asignación de aptitudes. No existen criterios definitivos para dar valores a esos parámetros, por lo que esta tarea depende en gran medida del buen criterio del programador. En principio, los algoritmos genéticos están reputados como técnicas de búsqueda *paramétricamente robustas*, es decir, funcionan —mejor o peor, pero funcionan— en un amplio rango de valores de sus parámetros. Sin embargo, las diferencias de eficiencia en la práctica son lo suficientemente espectaculares como para solicitar una mayor atención en dicha elección.

Se ha realizado una gran cantidad de pruebas (ver [DAVI91,p7], [MICH94,p70] y [PROC89]) sobre una amplia variedad de problemas de búsqueda y optimización con la intención de determinar rangos de buen funcionamiento para los parámetros del SGA. Aunque los resultados —medidos en términos de prestaciones— son dependientes en mayor o menor medida del problema en particular, se pueden extraer las siguientes conclusiones generales:

⋆ El tamaño de la población *PopSize* varía habitualmente entre 50 y 100 individuos; valores menores suelen plantear graves problemas de convergencia prematura y valores mayores requieren un gran esfuerzo computacional sin obtener mejoras apreciables.

⋆ El tamaño de los individuos *Len* suele venir dado como una consecuencia de los criterios de representación y codificación del problema. El único consejo que cabe dar es el de no alargar innecesariamente dicha cantidad salvo si se tiene la garantía de que al introducir redundancias se van a obtener prestaciones muy superiores.

⋆ Las mejores prestaciones se obtienen en general (ver [GREF90], [HUNT95], [LEVI95], [SGAC94] y [JANI93]) con tasas de cruce que varían entre el 20% y el 60% y tasas de mutación entre el 0.1% y el 5%, aunque no es raro ver otros valores. En lo que se refiere a otros operadores específicos, no existen criterios generales; tan sólo que los operadores de alteración (vg., la inversión, y en general todos los unitarios) se usan con mucha menor frecuencia que los de recombinación.

⋆ El máximo número de iteraciones depende de la precisión especificada y varía mucho de un problema a otro; sirvan como valores orientativos $MaxIter \sim 50$ para problemas de evaluación compleja y $MaxIter \sim 1000$ para problemas de evaluación sencilla. En cuanto a la tolerancia de error se hace notar que está fuertemente condicionada por la representación elegida y por la 'rugosidad' de la función de evaluación. Para problemas 'reales' de ingeniería se usan valores entre 10^{-2} y 10^{-5}.

⋆ Un valor típico para el periodo de actualización de la ventana de desplazamiento de las evaluaciones es $W = 5$ generaciones.

La robustez paramétrica garantiza el buen funcionamiento del AG clásico con esos valores para una gran variedad de problemas de optimización, pero ¿Qué ocurre cuando el problema no está dentro de esa "gran variedad"? Lo común es que una parametrización como la anterior proporcione mediocres prestaciones. El caso más habitual de esta situación se presenta cuando se usa un AG con una estructura apreciablemente distinta de la del SGA y/o con nuevos operadores genéticos.

3.2.8.1 Ajuste automático de los parámetros de un AG

Se han planteado varias técnicas para resolver el *problema de la parametrización de los AGs*. La más tosca es la de *búsqueda exhaustiva* o de 'fuerza bruta', que sólo se debería usar cuando las demás fallen o sea costosa su implantación; en los demás casos son preferibles técnicas de ajuste automático (o *autosintonizado*) de los parámetros.

En 1986 Grefenstette propuso ajustar los parámetros de un AG a través de otro ***algoritmo genético supervisor***, al cual se le da una parametrización típica. Esa estrategia sólo es útil para realizar estudios genéricos de la parametrización en una 'clase' de AGs, pero no para realizar la parametrización de un AG al resolver un problema concreto.

Posiblemente el procedimiento más práctico y que mejor se adapta a la filosofía de la computación evolutiva es el ***método de los parámetros evolutivos***, propuesto por Davis en 1989 (ver [DAVI91,p93]), y que fue mencionado al describir los cruces con adaptación (apdo. 3.2.6.1). La idea básica consiste en someter a los parámetros de aplicación de los operadores genéticos a un proceso de evolución análogo al de los AGs con la idea final de utilizar con más frecuencia los operadores que hagan un 'buen trabajo'. A cada operador se le asigna una *aptitud operacional* en función del incremento de aptitud que proporcione a la población su aplicación en un instante dado y se actualiza periódicamente. En [DAVI91,p94] se comentan los detalles de la implantación.

Nótese que el método de los parámetros evolutivos realiza un ajuste dinámico de los parámetros de un AG, de manera que en cada momento se tiene presumiblemente la mejor parametrización posible. Esto presenta el inconveniente de que un operador que sea importante únicamente en las fases finales de la evolución puede perderse en las primeras etapas; para evitarlo, se debe dar a los operadores con una probabilidad nula de aplicación una aptitud no nula que les permita volver a funcionar cuando el momento sea propicio. Obsérvese también que este método es también muy útil para probar nuevos operadores genéticos, pues en el propio proceso de evolución se irán desechando los que sean inservibles.

3.2.9 Incorporación del conocimiento específico en los AGs. Hibridación

Los algoritmos genéticos son, por construcción, métodos de búsqueda ciega: el proceso a optimizar es una caja negra que asigna a cada individuo una aptitud, resultando desconocidas para el algoritmo las interioridades del proceso. Es interesante esa opacidad en la medida en que proporciona algoritmos de propósito general y permite realizar la búsqueda con una información mínima; y efectivamente, los AGs proporcionan soluciones válidas a una amplísima variedad de problemas sin mayor requerimiento que la existencia de un criterio útil de idoneidad (función de aptitud). Esto tiene una contrapartida que a estas alturas resulta evidente por repetida: los algoritmos genéticos son métodos intrínsecamente débiles. Las consecuencias más inmediatas de la debilidad son la ineficencia y la poca flexibilidad en el tratamiento de los puntos no factibles. Tal es así que, para la mayoría de problemas reales de búsqueda que surgen en la ingeniería, los AGs 'básicos' son métodos esencialmente globales y excesivamente ineficientes como para ser competitivos, en el sentido de que a efectos prácticos no son mucho mejores que las búsquedas aleatorias. Como la debilidad es intrínseca, cualquier intento de mejora cualitativa ha de pasar por la incorporación, directa o indirecta, de conocimiento específico de cada problema. La incorporación de esa información no sólo origina inconvenientes teóricos —pérdidas de generalidad, de robustez, de soporte teórico, etc.— sino también inconvenientes prácticos, pues el conocimiento específico no siempre es fácil de incorporar, no se dispone del suficiente o no es el adecuado. Debe tenerse en cuenta que la incorporación de conocimiento específico incrementa fuertemente la sensibilidad al ruido de modo que, si no es muy fiable, el incremento de eficiencia será pequeño, y en el peor de los casos podría desorientar al algoritmo. Tampoco puede ser 'excesivamente específico' porque se incrementaría la tendencia a la convergencia prematura.

Pese a todos esas dificultades, la incorporación del conocimiento específico resulta inevitable en muchos problemas no triviales, y en todo caso recomendable cuando se disponga de él. En los AGs esa incorporación se ha hecho tradicionalmente mediante técnicas de *hibridación* que seguidamente se verán. Recientemente se han propuesto varias técnicas de incorporación directa del conocimiento específico en el propio AG; destaca entre ellas el paradigma de los Programas de Evolución de Michalewicz, que se estudiará con detenimiento en el próximo capítulo.

La idea general de las técnicas de ***hibridación*** consiste utilizar el AG para la búsqueda global y encargar la búsqueda local a métodos específicos, vg. de escalada —eso se expresa gráficamente diciendo que el AG se encarga de buscar los picos de aptitud y el escalador de treparlos—. En último término se trata de aprovechar las buenas propiedades de cada método.

La mejor manera de llevar a cabo la hibridación es de forma modular. La conexión entre módulos se puede hacer secuencialmente o en paralelo. En la *implantación secuencial* se deja que el AG converja suficientemente y entonces se ejecuta el método local sobre los mejores puntos obtenidos. En la *implantación en paralelo* varios procesadores subordinados realizan la búsqueda local y evalúan los individuos obtenidos y un procesador central conectado con todos ellos hace evolucionar la población proporcionada.

3.2.9.1 Técnicas genéticas de búsqueda local

En ocasiones sucede que no existen algoritmos heurísticos para resolver todas las búsquedas locales que se plantean en un procedimiento híbrido. Para tales subproblemas no queda más remedio que usar técnicas ciegas de búsqueda local. Como ya quedó dicho, los AGs son técnicas de búsqueda ciega global, pero no están especialmente dotados por sí solos para la búsqueda local. Existen varias maneras de corregir esa limitación; se expondrá aquí la más sencilla.

Goldberg [GOLD89,p202] fue el primero en notar que es posible obtener una información semejante a la proporcionada por el gradiente sin más que conmutar algún bit en los correspondientes individuos codificados; por eso propuso sustituir los procedimientos locales de escalada por otros que trabajen directamente sobre los individuos codificados. El modo de operación es así:

1. Se toman uno o más de los mejores individuos de la población actual.

2. Se conmutan uno por uno y sucesivamente todos los bits de los individuos interesados y se evalúan los individuos resultantes.

3. Se sustituyen los individuos originales por los mejores obtenidos en el paso anterior y se continúa la búsqueda genética.

Por motivos obvios a esta técnica se le dice de *emulación del gradiente*.

Capítulo 4

Extensiones de los Algoritmos Genéticos

En este capítulo se van a describir algunas extensiones del modelo básico de Algoritmos Genéticos. Todas ellas conservan los principios fundamentales de los AGs, en especial la opacidad y el paralelismo implícito, pero introducen cambios importantes en su estructura básica, lo que justifica su estudio por separado.

4.1 Algoritmos Genéticos con edades

Aunque no es conveniente tomar al pie de la letra la analogía con la evolución natural, resulta extraño no haber considerado un hecho incontestable: todos los organismos naturales nacen, se reproducen con mayor o menor éxito y más tarde o más temprano desaparecen. En 1994, Arabas, Michalewicz y Mulawka [MICH94,p70] propusieron introducir esta evidencia natural en los algoritmos genéticos de modo muy sencillo: en el momento de ser creado se le asigna a cada individuo una *duración* dependiendo de su aptitud y se pone a cero un contador de *edad*; tras cada generación se incrementa la edad de cada individuo y se eliminan aquéllos que hayan rebasado su duración.

Más allá de discusiones teóricas sobre la emulación de la naturaleza, la introducción de la edad en los individuos de los AGs está justificada por dos motivos concretos:

1. Ese mecanismo se basta para controlar la presión selectiva, de modo tal que hasta hace superflua la fase de selección (de criadores), lo que evita en principio todos los inconvenientes asociados a ella.

2. La introducción de la edad obliga a hacer variable el tamaño de la población, lo cual resulta ser una ventaja más que un inconveniente, pues proporciona un mecanismo de autorregulación de dicho tamaño.

Ambas cuestiones resultan ser del máximo interés: por un lado, se sabe que los mecanismos de selección en poblaciones finitas presentan un conjunto de limitaciones las cuales provocan una cascada de inconvenientes y disfunciones de no muy fácil solución; es razonable pensar que la sustitución del mecanismo de selección por otro no menos "natural" atenúe o incluso elimine gran parte de ésos inconvenientes. Por otro lado, el tamaño de la población es una de las elecciones más importantes a la hora de sintonizar un AG, pudiendo ser crítica en muchas aplicaciones: si el tamaño de la población es demasiado pequeño tendrá una acusada tendencia a la convergencia prematura, y si es demasiado grande seguramente desperdiciará recursos computacionales sin obtener mejoras apreciables. Por eso es conveniente estudiar las posibilidades de conseguir un mecanismo que adapte el tamaño de la población a las circunstancias particulares de la búsqueda.

Concretando términos, en los AGs con edades no hay selección y el reemplazo se limita a eliminar a los individuos que han sobrepasado su duración. Además, el tamaño de la descendencia está determinado por el tamaño actual de la población a través de

$$OffspSize[t] := \lfloor \rho \cdot Popsize[t] \rfloor$$

donde ρ es el *coeficiente de reproducción*, que toma valores próximos a 0.4. Cuanto mayor sea el valor de ρ mejores serán en general las prestaciones del AG, pero a costa de incrementar apreciablemente el esfuerzo computacional.

Por lo demás, la estructura del AG es la habitual, respondiendo al siguiente esquema básico:

```
BeginAlgorithm{AgedGeneticAlgorithm}
 AgesP = InitAges();
 P[0] = InitPop();
 FitP[0] = EvalPop(P[0]);
 for( t = 0; t < MaxNumGen; t++) {
  AgesP++;
  Q[t] = ChooseBreedersFrom(P[t], rho);   Selección al azar
  Q[t] = Crossover(Q[t]);
  Q[t] = Mutate(Q[t]);
  P[t] = Merge(P[t], Q[t]);
  FitP[t] = EvalPop(P[t]);
  P[t] = KillOldIndiv(P[t],FitP[t],AgesP);
  FitP[t] = RemoveKilledFit(P[t]);
  if( ChkTermCond(P[t], FitP[t]) )
   return;
 }
EndAlgorithm
```

Obsérvese que, como no hay selección propiamente dicha, todos los individuos de la población pueden ser elegidos como progenitores con idénticas probabilidades, independientemente de sus aptitudes.

A cada individuo se le asigna su duración al ser evaluado por primera vez y dicho parámetro permanece constante a lo largo de todo el proceso de evolución, hasta que el individuo desaparezca, lo cual sucede cuando y sólo cuando ha superado su duración. Por eso, el tamaño de la población tras la t-ésima generación es

$$Popsize[t+1] = PopSize[t] + OffspSize[t] - Killed[t]$$

donde $Killed[t]$ es el número de individuos que han rebasado su duración en dicha iteración.

El punto más delicado de estos métodos se refiere al criterio de asignación de duraciones a los individuos. Se comprende fácilmente que para ejercer un control eficaz sobre la presión selectiva es preciso que dicho criterio

(a) refuerce la presencia de los individuos más aptos y reduzca la presencia de los menos aptos y

(b) controle el tamaño de la población de modo que no sea tan pequeño que ocasione una pérdida de la diversidad ni tan grande que se desperdicie esfuerzo computacional o, peor aún, conlleve un crecimiento incontrolado de la población.

Tal y como se implantan los procesos de selección (inexistente) y reemplazo, es evidente que el tamaño de la descendencia de un individuo es directamente proporcional a su duración. Por eso, el primer requisito obliga a que se asignen las duraciones más altas a los individuos más aptos pues así estos individuos tendrán más posibilidades de formar parte de la descendencia. Sin embargo, no es conveniente asignar a los individuos una duración estrictamente proporcional a su aptitud, dado que se acrecienta la tendencia a violar el segundo requisito.

La práctica muestra que a la hora de calcular la duración de un individuo conviene considerar, además de su aptitud, el estado presente de la búsqueda, medido éste a través de parámetros tales como las aptitudes media y extremas de la población. Esencialmente, existen tres estrategias de caracter general para asignar la duración de un individuo:

1. ***Asignación proporcional***:

$$LifeTime(\mathbf{v}) = \min\{MaxLT, MinLT + DifLT\frac{Fitness(\mathbf{v})}{AvgFit}\}$$

2. ***Asignación lineal***:

$$LifeTime(\mathbf{v}) = MinLT + 2\,DifLT\frac{Fitness(\mathbf{v}) - MinFit}{MaxFit - MinFit}$$

3. ***Asignación bilineal***:

$$LifeTime(\mathbf{v}) = \begin{cases} MinLT + DifLT\dfrac{Fitness(\mathbf{v}) - MinFit}{AvgFit - MinFit} \\ \quad \text{sii} \quad Fitness(\mathbf{v}) \leq AvgFit \\ \frac{1}{2}(MinLT + MaxLT) + DifLT\dfrac{Fitness(\mathbf{v}) - AvgFit}{MaxFit - AvgFit} \\ \quad \text{sii} \quad Fitness(\mathbf{v}) > AvgFit \end{cases}$$

Los parámetros que acotan las duraciones, $MaxLT$ y $MinLT$, se dan a priori siendo sus valores por defecto 7 y 1 respectivamente. Por otra parte, se define

$$DifLT := \frac{1}{2}\left(MaxLT - MinLT\right)$$

La primera estrategia tiene cierto parecido con la selección por sorteo en el sentido de que las probabilidades de supervivencia de un individuo son proporcionales a su aptitud relativa. El tope en $MaxLT$ se añade para evitar el crecimiento descontrolado de la población.

La experiencia muestra que conviene considerar también la aptitud de los individuos referida no a la aptitud media, sino a la mejor aptitud obtenida hasta entonces; a esto se le conoce como *aptitud objetiva*:

$$ObjFitn(\mathbf{v}) := \frac{Fitness(\mathbf{v})}{MaxFitn(P)}$$

Esta observación motiva la estrategia de asignación lineal, con la cual se mejoran las prestaciones de la asignación proporcional.

La estrategia de asignación lineal presenta el inconveniente de que cuando hay muchos individuos con aptitudes similares a la mejor, se asignan largas duraciones a todos ellos, lo que hace crecer fuertemente el tamaño de la población y con él el coste computacional. Con la estrategia de asignación bilineal se trata de rebajar esta tendencia acentuando las aptitudes próximas a la mejor.

En [MICH94,p73] se comparan las prestaciones —en términos de coste computacional y precisión— del SGA clásico frente a una modificación en la que únicamente se introduce la edad. Los ensayos sobre cuatro problemas de optimización numérica mostraron que la estrategia de asignación lineal es la que proporciona mayor precisión, pero también la más costosa. La menos costosa es la bilineal, pero no tiene buena precisión. La asignación proporcional establece un punto de compromiso entre costes y precisión.

Al ejecutar un AG con edades la evolución del tamaño de la población $PopSize[t]$ presenta una forma oscilante muy característica. Michalewicz [MICH94,p74] lo explica así:

> Al principio, cuando la variedad de aptitudes es relativamente alta, el tamaño de la población crece. Esto quiere decir que el AG con edades realiza una búsqueda en anchura del óptimo. Una vez que ha sido localizada la vecindad del óptimo, el algoritmo comienza a converger y el tamaño de la población se reduce. Sin embargo, aún se sigue buscando una mejoría. Siempre que se presente una ocasión para obtener un mejor resultado, otra "explosión demográfica" tiene lugar, la cual es sucedida por otra etapa de convergencia. Al parecer, los AGs con edades incorporan un proceso de autoajuste adaptando el tamaño de la población a cada etapa del proceso de evolución.

Con independencia de bajo qué condiciones teóricas se pueda garantizar el correcto funcionamiento de ese mecanismo, es un hecho comprobado que su introducción reduce grandemente la sensibilidad ante el tamaño inicial de la población que sufrían los AGs sin edades: ya no importa el valor de $PopSize[0]$ pues el propio AG se encargará de ajustar convenientemente el tamaño en pocas generaciones. Eso es especialmente útil cuando se trabaja con problemas fuertemente restringidos. En ellos, dada la dificultad de encontrar soluciones factibles, se parte una población inicial de pequeño tamaño y se la hace evolucionar procurando mantener la factibilidad; según se vayan encontrando nuevas soluciones factibles el tamaño de la población se irá incrementando hasta alcanzar un valor óptimo.

Las perspectivas de los AGs con edades son excelentes; aparte del ya comentado control automático del tamaño de la población, parecen proporcionar mecanismos de selección más robustos que los directos. En el momento presente la investigación de los AGs con edades tiene tres frentes abiertos: estudio de sus prestaciones en una variedad de problemas lo más amplia posible, estudio de nuevas estrategias de asignación de duraciones y estudio teórico de la convergencia.

4.2 Algoritmos Genéticos con mecanismos de escisión

Los mecanismos de escisión se introducen en los AGs para separar grupos de individuos con similares capacidades para explotar ciertas características del entorno (ie., de la función de evaluación en cierta región del espacio de búsqueda).

El lugar natural para aplicar los mecanismos de escisión surge en problemas de búsqueda múltiple. Bajo el término de *búsqueda múltiple* se engloban dos tipos de problemas:

1. ***Búsqueda exhaustiva:*** Consiste en localizar todos los óptimos de una función objetivo, o bien todos los puntos que verifican cierto 'criterio amplio de idoneidad', vg., todos los subóptimos, todos los puntos que rebasen cierto umbral, etc.

2. ***Búsqueda multiobjetivo:*** Consiste en optimizar simultáneamente varios objetivos. Dicho de otra manera, se trata de optimizar una función objetivo vectorial. También se la conoce como *búsqueda multicriterio.*

Se estudiará por separado cómo se implanta la búsqueda genética múltiple en uno y otro caso.

Cuando se plantea resolver un problema de búsqueda exhaustiva a través de un algoritmo genético la primera idea que surge consiste en ejecutar varias veces el AG partiendo de poblaciones iniciales distintas, con la esperanza de que en cada ejecución se encuentre un óptimo distinto. Esta es la *técnica de iteración.* Se comprueba inmediatamente que si el problema posee p óptimos con igual verosimilitud de ser encontrados, el número esperado de ejecuciones independientes que debería realizarse asciende a $p \cdot (1 + 1/2 + \ldots + 1/p)$. Sin embargo, lo más común es que no todos los óptimos sean igual de fáciles de encontrar, en cuyo caso la estimación anterior del número de ejecuciones suele verse ampliamente rebasada. Este hecho limita grandemente la aplicabilidad de dicha técnica, de modo que en la práctica sólo se toma en consideración cuando se dispone de computadores vectoriales en los que puedan evolucionar simultáneamente varias subpoblaciones de modo independiente.

Un enfoque más natural para resolver problemas de búsqueda exhaustiva propone estimular la formación de ***subpoblaciones estables*** en torno a cada uno de los óptimos, de modo que según progrese el AG más cerca se encuentren los individuos de su respectivo óptimo. Obsérvese que si no se toman precauciones, la búsqueda genética tiende a converger a uno de los óptimos: el mejor si tienen distintas aptitudes o a uno cualquiera si las aptitudes son las mismas (por deriva genética).
Esencialmente existen dos maneras de implantar esa propuesta: por agregación y por segregación. Ambas hacen uso de la idea de *aptitud compartida*, que a su vez se basa en la de *distancia* entre individuos, medida esta bien sea a nivel de genotipo bien a nivel de fenotipo. Fundamentalmente se trata de plasmar numéricamente el hecho de que la aptitud de un individuo se ve reducida al compartir los recursos de su entorno con sus vecinos. Concretamente, la aptitud compartida de cierto individuo $\mathbf{v}$ se evalúa así:

$$ShFitness(\mathbf{v}) := \frac{Fitness(\mathbf{v})}{\sum_{w \in P} sh(dist(\mathbf{v}, \mathbf{w}))}$$

Las funciones de compartición, $sh(d)$, deben verificar las siguientes condiciones con las que se garantiza la estabilidad de las subpoblaciones:

- $0 \leq \mathrm{sh}(d) \leq 1 \qquad (\forall d \in \mathcal{R}^+)$
- $\mathrm{sh}(0) = 1$
- $\lim_{d \to \infty} \mathrm{sh}(d) = 0$

Con esta modificación, la aptitud del individuo **v** decrecerá proporcionalmente al número y a la cercanía de los puntos vecinos, de manera que cuando haya muchos individuos en la misma vecindad, los unos reducirán la aptitud neta de los otros, y si un individuo está aislado su aptitud neta prácticamente no disminuirá. El resultado final es que se fuerza a mantener la diversidad impidiendo el crecimiento incontrolado de una subpoblación a expensas de las demás.

4.2.1 Métodos de formación de subpoblaciones estables por agregación

Los ***métodos de agregación*** consisten en definir una función de compartición y hacer evolucionar un AG clásico cuyo tamaño de población sea preferiblemente un múltiplo del número de óptimos. Hay muchas posibilidades para elegir la función de compartición, Goldberg y Richardson [PROC89,p43] proponen la siguiente:

$$sh(d) := \begin{cases} 1 - \left(\dfrac{d}{\sigma_{sh}}\right)^{\alpha} & \text{sii} \quad d < \sigma_{sh} \\ 0 & \text{sii} \quad d \geq \sigma_{sh} \end{cases}$$

donde α es un parámetro con el que se regula la densidad de la subpoblación en función de la distancia a su 'centro', y σ_{sh} es una cantidad que mide la máxima distancia entre cadenas necesaria para formar tantas subpoblaciones como óptimos. El cálculo de σ_{sh} depende de cómo se haya definido la distancia entre individuos; en [PROC89,p43] se detalla cómo se realiza ese cálculo para elecciones típicas de la distancia.

Este enfoque adolece de al menos dos defectos: los cálculos de las aptitudes compartidas consumen gran cantidad de recursos computacionales y además no siempre se puede garantizar que *PopSize* sea un múltiplo del número de óptimos, por la sencilla razón de que ese número suele ser desconocido a priori. Para evitarlos, Beasley, Bull y Martin [MICH94,p177] propusieron una técnica alternativa, que en cierta medida es una mezcla de las dos vistas hasta ahora. Se la llama técnica de ***agregación sucesiva***, y se basa en la siguiente idea: siempre que se encuentre un óptimo se modifica, para la siguiente ejecución, la función de evaluación con el fin de disuadir al AG de volver a descubrir ese mismo óptimo otra vez. La modificación de la función de evaluación se lleva a cabo considerando las distancias de los individuos de la población al óptimo recién hallado, de modo similar a como se acaba de describir. En cierta medida, se trata de que los sucesivos ensayos que vaya realizando el AG incorporen el conocimiento adquirido en ensayos anteriores.

4.2.2 Métodos de formación de subpoblaciones estables por segregación

Los *métodos de segregación* o ***especiación*** consisten en crear *especies* en la población de manera que sólo se puedan cruzar individuos de la misma especie. La implantación se lleva a cabo etiquetando los individuos e introduciendo el *emparejamiento restrictivo*, esto es, impidiendo los cruces entre individos de distintas especies. La clave de estos métodos consiste en elegir un criterio de etiquetado y la correspondiente función de compartición de modo tal que cada especie ocupe un óptimo.

Una generalización de este modelo prohibe o permite los cruces entre individuos según la especie a que pertenezca cada uno; aparte de mejorar la diversidad, se trata de aprovechar las ventajas proporcionadas por la especialización y la cooperación entre especies. Por motivos obvios, a estos métodos se les llama ***métodos de diferenciación***. El ejemplo más claro de diferenciación en la naturaleza es la de los organismos sexuados.

Debe quedar claro que el uso de las funciones de compartición no es exclusivo de los problemas de optimización exhaustiva. Son recomendables siempre que se precise forzar el mantenimiento de la diversidad, por ejemplo, al optimizar funciones fuertemente multimodales, o al hibridar AGs con métodos de búsqueda local (ver sección 3.2.9).

4.2.3 Aplicación de los mecanismos de escisión en búsquedas multiobjetivo

El otro lugar natural donde aplicar los mecanismos de escisión es en los problemas de búsqueda multiobjetivo, también llamada *multicriterio*. En muchos problemas prácticos, especialmente en los de toma de decisiones, es necesario hacer óptimos simultáneamente varios objetivos $f_1(\mathbf{x}), \ldots, f_r(\mathbf{x})$, o por lo menos resulta más realista lograr un "compromiso de optimidad" entre todos ellos. Por ejemplo, en una compañía comercial la maximización de beneficios como único objetivo no es una buena política a largo plazo, pues también se debe mirar por el máximo incremento de la cuota de mercado, del prestigio de la compañía, de su capacidad de maniobra en un mercado cambiante, etc. Se deben incluir todos esos factores en el modelo dado que no necesariamente son máximos cuando lo es el beneficio; el problema radica en que todos esos factores son difícilmente traducibles a costes y, en consecuencia, no se pueden englobar en un sólo objetivo.

Matemáticamente se trata de resolver el siguiente problema:

$$\left\{ \begin{array}{rll} \text{maximizar} & f_k(\mathbf{x}) & (\forall k = 1, \ldots, r) \\ \text{sujeto a} & \mathbf{x} \in \mathcal{X} \subseteq \mathcal{R}^m & \end{array} \right.$$

Hay ocasiones en que los objetivos $f_k(\mathbf{x})$ admiten una base común de medidas (por ejemplo, costes o beneficios equivalentes). En tal caso no es necesario recurrir a métodos de búsqueda multiobjetivo ya que se puede reducir el problema a uno uniobjetivo a través de una *función combinada de objetivos* del estilo de:

$$f(\mathbf{x}) := w_1 f_1(\mathbf{x}) + \ldots + w_r f_r(\mathbf{x})$$

donde los w_k son los factores de ponderación con los que se transforman todos los objetivos a la base común. Por conveniencia se suele hacer que la suma de los factores de ponderación valga 1. Es importante realizar una elección adecuada de los w_k, puesto que distintos valores a esos factores pueden proporcionar muy distintas soluciones al problema, y lo que es peor, una inapropiada elección suele proporcionar soluciones mediocres que no son buenas respecto de ningún objetivo.

Desafortunadamente, en muchos problemas prácticos sucede que los objetivos son difícilmente reducibles a una base común, tanto en sentido teórico (¿qué significado tiene traducir a beneficios la capacidad de maniobra de una empresa?) como práctico (¿Cómo se asignan valores apropiados a los factores de ponderación?), por lo que este enfoque resulta poco útil.

El primer intento de resolver problemas de optimización multiobjetivo utilizaba una población dividida en subpoblaciones de igual tamaño, cada una de las cuales era responsable de un único objetivo; la selección era interna (dentro de cada subpoblación) y el cruce se realizaba entre subpoblaciones. Todos los problemas de este tipo manifiestan una tendencia más o menos acusada a proporcionar soluciones mediocres[1], por lo que es necesario habilitar mecanismos adicionales que la corrijan (ver [MICH94,p179] y [GOLD89,p199]). Se pueden mejorar esos resultados introduciendo el concepto de ***soluciones no dominadas*** u *óptimos de Pareto*. Se dice que una solución $\mathbf{x} \in \mathcal{X}$ es un óptimo de Pareto cuando no existe ninguna solución mejor en ningún objetivo, es decir:

$$f_k(\mathbf{y}) \leq f_k(\mathbf{x}) \qquad (\forall k = 1, \ldots, r) \quad (\forall \mathbf{y} \in \mathcal{X},\ \mathbf{y} \neq \mathbf{x})$$

Presumiblemente todas las soluciones de interés estarán entre ellas.

La idea consiste en ordenar los individuos en función de su grado de dominio en la población: a los individuos no dominados se les da el primer puesto, el segundo se da a los individuos no dominados por otros individos más que por los del primer puesto, el tercero a todos los no dominados más que por los del primer y segundo puesto, y así sucesivamente. Esos índices se usarán como aptitudes brutas de los respectivos individuos, de modo análogo a como se hacía en 3.2.5.2 al describir los métodos de escalado basados en el orden. Las técnicas de escisión se introducen para mantener un alto grado de diversidad a lo largo de todo el proceso, pues sólo así se obtienen buenas soluciones.

[1]Es decir, para poder satisfacer más o menos todos los objetivos, se elige un punto que no es especialmente malo en ninguno de ellos, pero tampoco especialmente bueno.

4.3 Algoritmos Genéticos paralelos

Posiblemente la propiedad más importante de los AGs sea el paralelismo implícito. Ello no obsta para que sea del máximo interés realizar una implantación explícitamente paralela aprovechando las facilidades que da la propia estructura del AG. No existe un modelo típico de implantación en paralelo de un AG, entre otras cosas porque hay una gran variedad de estructuras de procesado en paralelo. A grandes rasgos, los modos de implantar un AG en paralelo responden a tres modelos:

1. ***Modelos centralizados:*** En ellos un proceso central coordina el funcionamiento de varios procesos subordinados. La implantación es directa y fácil de realizar: el proceso central se encarga de controlar los operadores de selección y reproducción, mientras que los subordinados únicamente evalúan aptitudes. Estos modelos presentan los dos inconvenientes inherentes al centralismo, a saber: antes que nada, es necesario que los tiempos de evaluación de los procesos subordinados sean análogos (sincronía), pues en caso contrario se producirán gran cantidad de tiempos muertos; por otra parte, el esquema centralizado presenta poca fiabilidad al depender críticamente del buen funcionamiento del proceso central.
   El requerimiento de sincronía se puede suavizar algo eligiendo los operadores adecuados para que el proceso central pueda trabajar con los individuos según van siendo evaluados.

2. ***Modelos distribuidos:*** En ellos un conjunto de procesadores idénticos realizan las operaciones genéticas y las evaluaciones independientemente los unos de los otros, pero accediendo a una zona común de *memoria compartida*. No existe mayor requisito de sincronía que el de que no se produzcan accesos simultáneos a la memoria compartida. A efectos externos, estos modelos funcionan de la misma manera que los centralizados, pero ahora la fiabilidad se ha incrementado grandemente, pues basta que funcione uno sólo de los procesadores para que el AG pueda realizar procesamiento útil.

3. ***Modelos en red:*** En ellos un conjunto de procesadores independientes ejecutan un AG convencional usando cada uno su propia zona de memoria, operadores genéticos y evaluaciones. Además, se establece una *topología de comunicación* entre los procesadores que permita el intercambio coordinado de información entre poblaciones con el fin de apoyar la búsqueda de mejores soluciones. Habitualmente dicho intercambio se lleva a cabo mediante técnicas de *migración* —ie., movimiento de individuos de una subpoblación a otra— y cruce entre individuos de distintas poblaciones.

No es infrecuente hibridar los mecanismos de procesamiento paralelo de AGs con otras técnicas heurísticas, con el fin de mejorar no sólo la velocidad sino también la precisión de la búsqueda (ver sección 3.2.9).

En cualquier caso, resulta muy sencillo y natural implantar un AG en paralelo, por lo que no es difícil pronosticar que éstas implantaciones se impondrán en cuanto se generalicen los procesadores en paralelo de bajo coste. Además, los nuevos procesadores masivamente paralelos permitirán la creación de nuevos y más complejos modelos de evolución artificial.

4.4 Algoritmos Genéticos diploides

Se ha comentado repetidamente que el hecho de obligar a que los genotipos respondan a la forma de cadenas binarias responde a la necesidad de obtener la máxima capacidad de procesamiento de esquemas. No obstante, es posible añadir nuevas características al genotipo sin perder por ello capacidad de procesamiento. Tal es el caso de los AGs con representación duplicada, o usando la terminología biológica, *diploides*. La idea está tomada de los organismos naturales y consiste en hacer que en cada gen haya lugar para dos alelos, de los cuales sólo uno, llamado *dominante*, se *expresará* a la hora de calcular la aptitud de cada individuo; en la reproducción, cada individuo hereda en cada gen un alelo al azar de cada uno de sus progenitores.

Como es habitual, la introducción de este mecanismo no responde tanto a la consecución de una mejor emulación de la naturaleza —lo cual está fuera del propósito general de la computación evolutiva— como a la obtención de unas propiedades concretas tendentes a mejorar la búsqueda genética. La analogía con la naturaleza sólo tiene sentido en la medida en que ilustre los procedimientos actuales o guíe el desarrollo de otros nuevos. En el presente caso, Goldberg [GOLD89,p149] propone la introducción del diploidismo en los AGs basándose en la siguiente analogía:

> Un ejemplo muy conocido de la memoria a largo plazo introducida por los mecanismos de diploidismo y dominancia se encuentra en los desplazamientos del equilibrio de poblaciones de la polilla moteada en Gran Bretaña durante la Revolución Industrial. La variedad silvestre (y originalmente la dominante) de este lepidóptero poseía alas blancas con pequeñas motas negras. Anteriormente a la Revolución Industrial, esa coloración era un camuflaje eficaz contra los pájaros y otros predadores del habitat natural de la polilla, los árboles cubiertos de musgo. A mediados del siglo XIX se capturaron variedades negras en los alrededores de ciudades industriales. Cuidadosos experimentos llevados a cabo por Kettlewell mostraron que la variedad moteada tenía ventaja en el entorno primitivo, mientras que la variedad negra tenía ventaja en el entorno industrial en el cual la contaminación había exterminado el musgo que cubría los troncos de los árboles. Sucedía que el grado de oscuridad está controlado por un

> único gen dominante, lo que indicaba que se había producido un desplazamiento de la dominancia. Cuando el equilibrio se desplazó hacia la variedad oscura, ésta se convirtió en dominante y la moteada en recesiva. Nótese que la variedad oscura no era una nueva invención; no era un caso de mutación fortuita que milagrosamente proporcionó la variedad correcta. En lugar de eso, la variedad negra había sido inventada antes, quizá como respuesta a bosques en los que el musgo había desaparecido por causas naturales. Cuando los residuos de la industria originaron la desaparición del musgo, la variedad oscura se reprodujo con mayor facilidad y evolucionó hasta convertirse en la variedad dominante. Con esta solución alternativa en la reserva, la polilla moteada pudo fácilmente adaptarse con rapidez a las presiones selectivas de su entorno cambiante.

En definitiva, la duplicación de los alelos introduce un mecanismo explícito con el que se evita la destrucción prematura de información que pudiera resultar valiosa en el futuro. Eso es especialmente interesante cuando se trabaja con funciones de aptitud cambiantes en el tiempo.

Fácilmente se observa que el punto crítico de los AGs diploides (en general, poliploides) radica en la elección de un criterio apropiado de dominancia, esto es, dados dos alelos de cierto gen ¿cuál de ellos se expresa?. En [GOLD89,p152] se discuten varios criterios de dominancia para algoritmos genéticos simples con genes que ocupan una y dos posiciones; también se comparan los resultados obtenidos al resolver mediante un SGA haploide y otro diploide un problema de la mochila en el que los coeficientes w_i varían periódicamente.

Los resultados teóricos y prácticos muestran que estos mecanismos son muy útiles para búsquedas en entornos no estacionarios, especialmente cuando hay alguna periodicidad. No obstante aún queda mucho por investigar en este campo, en particular se está dedicando gran esfuerzo a la determinación de criterios útiles de dominancia lo suficientemente generales.

Capítulo 5

Los Programas de Evolución

Los Algoritmos Genéticos fueron específicamente diseñados para proporcionar métodos de búsqueda de propósito general. Debido a ello, se construyen y ejecutan utilizando la mínima cantidad de información específica sobre el problema, a saber, una función de aptitud que evalúa cadenas binarias que representan a candidatos a solución. Esto se resume diciendo que los AGs son métodos de búsqueda ciega y codificada.
Esas exigencias mínimas proporcionan una gran generalidad, próxima a la de los métodos de búsqueda aleatoria. La generalidad es deseable tanto en cuanto reduce el esfuerzo de implantación de un algoritmo de resolución: cuanto más general sea un método menos costará implantarlo para resolver un problema en concreto. Esa generalidad, junto con la hipótesis de los bloques constructivos, hace de los AGs unos métodos de búsqueda muy eficaces, en el sentido de que dan resultados aceptables para una gran variedad de problemas. Además, la propiedad de paralelismo implícito confiere a los AGs una fortaleza (ie., eficiencia) superior a la de otros métodos de búsqueda ciega y en consecuencia una notable robustez. Debe considerarse esa eficiencia añadida solamente en términos relativos: los AGs siempre son mucho menos eficientes que los métodos heurísticos. Por eso se dice que los AGs son métodos de búsqueda intrínsecamente débiles.
La debilidad ocasiona múltiples inconvenientes prácticos a los AGs (ver apdo. 3.1.1); además, la debilidad unida a la rigidez de la representación hace que los AGs tengan serias dificultades para considerar restricciones no triviales del espacio de búsqueda. Esto es especialmente grave en problemas prácticos de "ingeniería real" en los cuales las restricciones ocupan un lugar fundamental.

Como ya se indicó en el apartado 3.2.9, al ser la debilidad de los AGs intrínseca cualquier intento de mejora cualitativa debe pasar por la incorporación, directa o indirecta, de información específica; se comentaban allí también las dificultades para obtener dicha información, así como los inconvenientes de orden teórico y práctico que conlleva. Para resolverlos se proponía una técnica de incorporación indirecta del conocimiento específico hibridando los AGs con técnicas heurísticas de búsqueda local. Evidentemente, la aplicabilidad de esta metodología queda supeditada a la existencia de procedimientos de búsqueda local.

En 1994 Michalewicz propuso una estrategia de incorporación directa del conocimiento específico con la que se atacaba el problema por su base: la representación. Se exige en los AGs la representación a través de cadenas binarias con el fin obtener la máxima generalidad y el máximo grado de paralelismo implícito (ver apdo. 3.2.1); sin embargo, esa representación es tan rígida en la práctica que merece la pena considerar otras más flexibles aunque con ello se pierda algo de generalidad y de paralelismo implícito. A fin de cuentas, una 'representación directa' tan sólo precisa información genérica, por lo que las consecuencias prácticas de la pérdida de generalidad no serán muy grandes; por otra parte, existe la posibilidad de compensar la pérdida de paralelismo implícito procesando más información útil, esto es, de compensar la violación del *principio del alfabeto de símbolos mínimo* dando más importancia al *principio de los bloques constructivos con significado.*

A las implantaciones que resultan de ese enfoque se les llama ***Programas de Evolución*** (en lo sucesivo, PEs) y comparten punto por punto la estructura básica de los AGs, distinguiéndose de éstos solamente en el nivel interno por el uso de:

(a) una representación lo más *natural* posible, ie., próxima al dominio del problema y que a la vez sea sea *útil* ie., significativa para el problema que se va a resolver. En la práctica eso se consigue utilizando estructuras de datos adecuadas al problema (vg. vectores reales, matrices, listas, conjuntos, árboles, etc.)

(b) operadores genéticos específicos *sensibles al contexto*, ie. adaptables al dominio, y lo más *cerrados* posible.

Si se hace adecuadamente esta 'incorporación directa de conocimiento específico en el AG' se obtienen procedimientos de búsqueda que pueden competir con éxito en términos de eficiencia y capacidad de tratamiento de restricciones con otros procedimientos fuertemente heurísticos (y por tanto, con altos costos de implantación).

En la gráfica de generalidad frente a eficiencia (ver figura 5.1 y el apdo. 1.2.3) los AGs se sitúan en la zona más alta de la región ocupada por la computación evolutiva, mientras que los PEs se sitúan en una amplia región del centro (alta eficiencia). En el extremo derecho (máxima eficacia), se sitúan las estrategias evolutivas, que serán tratadas en el próximo capítulo.

Un par de ejemplos ilustrarán mejor lo discutido: Los PEs surgieron al resolver problemas de optimización paramétrica en los que se requería alta precisión; en tal circunstancia la representación a través de enteros binarios obliga a utilizar individuos de tamaño muy grande, lo que incrementa fuertemente los requerimientos computacionales y de almacenamiento. No acaban ahí los problemas, pues

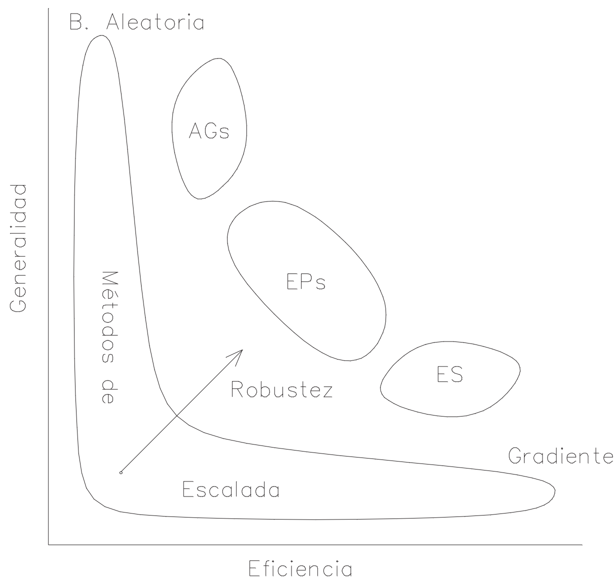

Figura 5.1: Generalidad vs. Eficiencia

la experiencia muestra que los operadores genéticos clásicos son apreciablemente ineficaces para realizar el 'ajuste fino' en las etapas finales de la evolución. Por unos y otros motivos, la aplicación de los AGs a este tipo de problemas da resultados francamente mediocres. Ahora bien, estos resultados mejoran sensiblemente (ver [MICH94,p95]) si se usa una representación y unos operadores genéticos 'naturales', es decir: los individuos son simplemente vectores de números en coma flotante en los que cada componente es asimilable a un gen, la mutación altera levemente el valor de una de las componentes del individuo y el cruce proporciona dos individuos que 'promedian' de alguna manera los valores de sus progenitores. Esta mejora en las prestaciones es sorprendente, habida cuenta de que el nuevo alfabeto de símbolos ha reducido grandemente la variedad de esquemas, y con ella la eficiencia del procesamiento en paralelo. Ese fenómeno se explica porque la adición de significado a los genotipos y a los operadores ha mejorado la capacidad de procesamiento de esquemas útiles, hasta el punto que compensa sobradamente la pérdida de variedad de esquemas. Nótese que la incorporación de conocimiento específico ha sido mínima.

El siguiente ejemplo se ha extraído de un problema de ingeniería real, concretamente de un proyecto de I+D que está siendo desarrollado para la empresa IBERDROLA (ver [BERN95]). A grandes rasgos, el problema consiste en lo siguiente: dados un conjunto de estaciones generadoras de potencia eléctrica, otro conjunto de nodos consumidores de potencia y un conjunto $\mathcal{L}$ de 'líneas viables' que los comunican entre sí, se trata de hallar la configuración de la red que minimiza el coste total de la instalación, entendido este como la suma de los costes fijos de amortización y mantenimiento de cada rama más los costes asociados a las pérdidas de potencia activa en cada una (ver figura 5.2).

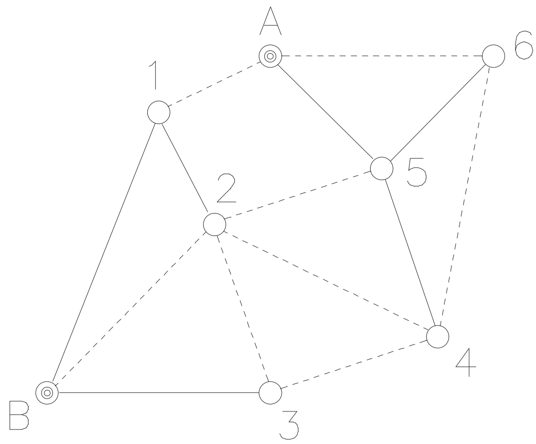

Figura 5.2: Red de distribución de potencia

Matemáticamente se expresa así:

$$\underset{V_{ij}\in\mathcal{V}}{\text{minimizar}} \sum_{(ij)\in\mathcal{L}} CF_{ij}\cdot Y_{ij} + CV_{ij}\cdot P_{ij}$$

sujeto a las siguientes condiciones:

— Las Y_{ij} son variables categóricas de selección. $Y_{ij} = 1$ indica que la rama (ij) está incluída en la configuración actual, mientras que $Y_{ij} = 0$ indica que no está incluída; en tal circunstancia, la respectiva potencia P_{ij} valdrá necesariamente 0.

— Por motivos técnicos no se permiten bucles en la configuración, $\mathcal{L}$, es decir, el dominio del problema es el de las configuraciones con estructura de árbol (radiales, en la jerga al uso).

— La red debe verificar las ecuaciones de Kirchhoff en forma de potencias (ecuaciones de flujos de carga), teniendo en cuenta que las potencias activas en los nudos consumidores están especificadas.

— Además, para que la red funcione correctamente, las tensiones en todos los nudos no deben rebasar ciertas tolerancias alrededor de sus puntos de trabajo nominales ($V_{ij} \in \mathcal{V}$) y los factores de potencia en los nudos consumidores no deben bajar de cierto valor mínimo (habitualmente 0.85).

Ahora el problema no está en el alfabeto de símbolos, ya que el alfabeto binario sigue siendo el más apropiado para indicar la presencia o ausencia de una línea en la configuración. El problema está en que la representación mediante cadenas binarias no es apropiada para considerar las restricciones anteriores. En efecto, si se usa una estructura de cadena binaria deben habilitarse mecanismos para distinguir las cadenas factibles de las que no lo son (ya sea por no ser radiales o por ser incompatibles con las especificaciones eléctricas); además, los operadores genéticos tradicionales no conservan la factibilidad, de hecho suelen destruirla por ser el problema fuertemente restringido. Si se usa el enfoque de los PEs la representación más natural de las configuraciones es utilizando estructuras de árboles; de esta manera resulta sencillo modificar los operadores genéticos para que conserven dichas estructuras y, en general, resulta mucho más sencillo verificar las condiciones de factibilidad, por estar el espacio de búsqueda más próximo al dominio del problema.

Debe quedar clara la diferencia entre los PEs y los AGs con codificadores (ver apdo. 3.2.2): los AGs con decodificadores aproximan el espacio de búsqueda X al dominio $\mathcal{X}$ a través de la codificación, pues la representación sigue siendo a través de cadenas binarias; los PEs eluden en la medida de lo posible la codificación y trabajan directamente sobre el dominio del problema, aunque ello conlleve añadir conocimiento específico a la representación.

Dada la gran variedad de representaciones posibles, no es fácil hacer una clasificación exhaustiva de los PEs; sin embargo, sí que se pueden distinguir dos grandes grupos de PEs según la naturaleza de los objetos tratados. Se habla así de PEs especializados en optimización numérica y PEs especializados en optimización combinatoria (ver apdo. 1.2.1); dentro de su variedad, las representaciones y operadores utilizados en cada grupo son radicalmente distintos del otro. En las páginas que siguen se describe un ejemplo característico de cada uno de ellos; se han elegido ejemplos lo suficientemente sencillos como para mostrar claramente el modo de operación y lo suficientemente completos como para dar una visión general de cómo son los PEs de cada tipo.

5.1 Programas de Evolución especializados en optimización numérica

Ya se ha comentado que los AGs no son especialmente eficientes al realizar optimizaciones numéricas. A las dificultades genéricas de los AGs se suman otras específicas que se resumen en la incapacidad para proporcionar resultados lo suficientemente precisos (ajuste fino local). Esa incapacidad es inherente al uso de una representación mediante cadenas binarias, por lo que la utilización de codificaciones distintas de la habitual no proporciona más que mejoras superficiales. Múltiples resultados experimentales avalan la conveniencia de usar una representación y unos operadores más próximos al dominio del problema. Concretamente, para problemas de optimización paramétrica se prefiere usar la representación directa, ie., cada candidato a solución se representa como un vector de números en coma flotante (abreviadamente, ***representación FP***). Esa representación proporciona al menos tres ventajas prácticas:

1. La notación FP cuantifica el espacio de búsqueda mediante una rejilla logarítmica, lo cual incrementa la capacidad de ajuste fino local.
2. Al ser más natural, permite simplificar el tratamiento de las restricciones.
3. Por el mismo motivo, es más fácil añadir más conocimiento específico a través de operadores apropiados.

Hay una gran variedad de operadores que se pueden aplicar con esa representación. Seguidamente se describen los más utilizados.

Mutaciones numéricas

Considérese que en la t-ésima generación de la evolución de un PE numérico se ha elegido para ser mutado el j-ésimo gen de cierto individuo $\mathbf{x}[t] = (x_1, \ldots, x_j, \ldots, x_m)$. Se denotan como u_j y l_j a las cotas del dominio de la correspondiente variable.

Mutación numérica estacionaria: En el j-ésimo gen se introduce un valor aleatorio URand$[u_j, l_j]$. Se parece a la mutación binaria clásica, sólo que se aplica sobre los genes, no sobre los bits.
Aplicado tal cual éste es un operador bastante drástico, por eso se suele emplear una versión suavizada en la que la componente x_j se altera así:

$$x_j[t+1] \leftarrow r \cdot x_j[t]$$

donde r es un número aleatorio gaussiano de media 1 y dispersión típica previamente especificada. También se usan a veces otras distribuciones de probabilidad.

Mutación numérica no estacionaria: Es más apropiada que la anterior cuando se usa una representación FP. Consiste en hacer

$$x_j[t+1] \leftarrow \mathrm{flip}\Big(x_j[t] + \Delta\left(t, u_j - x_j[t]\right), x_j[t] - \Delta\left(t, x_j[t] - l_j\right)\Big)$$

donde	u_j y l_j	son las cotas del dominio de la j-ésima variable
	flip(p,q)	es una función aleatoria que devuelve p o q al azar con un 50% de probabilidades
	$\Delta(t, z)$	es una función aleatoria que devuelve un valor en el intervalo $[0, z)$ de modo tal que la probabilidad de que $\Delta(t, z)$ sea próxima a cero se va incrementando según crece t.

La forma concreta de $\Delta(t, z)$ depende del problema. No obstante, las elecciones más comunes en problemas de optimización numérica son:

$$\Delta(t, z) := \left(1 - r^{(1-t/T)^b}\right) \cdot z \qquad \text{o bien} \qquad \Delta(t, z) := r\,(1 - t/T)^b \cdot z$$

siendo $r \leftarrow$ URand[0,1]
$T = MaxIter$
y b un parámetro que regula el grado de no uniformidad. Un valor habitual es $b = 5$.

De este modo se consigue que la mutación haga un recorrido uniforme del especio de búsqueda en las primeras etapas y después una búsqueda más local (ajuste fino), puesto que se incrementa la posibilidad de generar un número muy próximo a su antecesor.

La introducción de un operador de mutación no estacionaria adecuadamente parametrizado tiene efectos altamente beneficiosos en los PEs especializados en optimización numérica. No sólo se incrementa notablemente la precisión del algoritmo, sino que por lo general también mejora la velocidad de convergencia.

A veces se aplica también la mutación no estacionaria sobre todas las variables del vector $\mathbf{x}[t]$. Así se obtiene en cada mutación un vector completamente distinto en las primeras etapas, y cada vez más parecido al original al progresar la evolución.

Existen ciertos problemas de optimización numérica en los que la dirección de búsqueda está bastante clara pero se encuentra llena de 'obstáculos' en los que el algoritmo se puede atascar; el caso más común se da en el cálculo de trayectorias para el pilotado automático de vehículos. La mutación no estacionaria proporciona un mecanismo válido con el que sortear esos obstáculos sin más que cambiar la forma de la función $\Delta(t, z)$: según va creciendo t se hace que $\Delta(t, z)$ se aproxime a z en vez de a 0; así los cambios introducidos por la mutación van siendo cada vez más drásticos, lo que permite salir del atasco.

Cruces numéricos

Considérese que en la t-ésima generación de la evolución de un PE numérico se han elegido para ser cruzados los individuos $\mathbf{x}[t] = (x_1, \ldots, x_m)$ e $\mathbf{y}[t] = (y_1, \ldots, y_m)$.

Cruce simple: Análogo al cruce binario clásico. En este caso la posición de corte siempre se elige entre genes.

Cruce aritmético: Realiza una combinación lineal de los progenitores así:

$$\begin{array}{ccc} \mathbf{x}[t] & & a \cdot \mathbf{x}[t] + (1-a) \cdot \mathbf{y}[t] \\ & \Longrightarrow & \\ \mathbf{y}[t] & & a \cdot \mathbf{y}[t] + (1-a) \cdot \mathbf{x}[t] \end{array}$$

El valor del parámetro a depende del problema, pudiendo ser constante (cruce aritmético estacionario) o variable con la edad de la población (cruce aritmético no estacionario).
De acuerdo con [MICH94,p156] este tipo de cruce proporciona estabilidad a las búsquedas numéricas.

Cruce aritmético sencillo: Consiste en aplicar el cruce aritmético no sobre los vectores completos, sino sobre una sóla componente.

Cruce mixto: Es una mezcla del cruce numérico simple y del cruce aritmético. Su modo de operar se explica mejor gráficamente:

$$\begin{array}{ccc} (\mathbf{x}_I[t] \mid \mathbf{x}_D[t]) & & (\mathbf{x}_I[t] \mid a \cdot \mathbf{y}_D[t] + (1-a) \cdot \mathbf{x}_D[t]) \\ & \Longrightarrow & \\ (\mathbf{y}_I[t] \mid \mathbf{y}_D[t]) & & (\mathbf{y}_I[t] \mid a \cdot \mathbf{x}_D[t] + (1-a) \cdot \mathbf{y}_D[t]) \end{array}$$

Cruce heurístico: Este operador presenta dos singularidades que lo distinguen de los otros cruces:

- Utiliza explícitamente valores de la función objetivo, de ahí su nombre.
- Produce a lo sumo un sólo descendiente, pudiendo darse el caso de que no genere ninguno.

El modo de operación es éste: dados dos progenitores $\mathbf{x}_1$ y $\mathbf{x}_2$ en los que, por convenio, $\mathbf{x}_1$ es mejor que $\mathbf{x}_2$ (abreviadamente, $\mathbf{x}_1 \succ \mathbf{x}_2$), el candidato a descendiente se obtiene así

$$\mathbf{x}_3 = r \cdot (\mathbf{x}_1 - \mathbf{x}_2) + \mathbf{x}_1$$

donde r es un número aleatorio entre 0 y 1.
Si $\mathbf{x}_3$ es factible se confirma como descendiente, en caso contrario se genera otro número aleatorio r y con él otro candidato a descendiente. Si al cabo de cierto número de intentos no encuentra un cadidato factible, el operador da por terminada la búsqueda sin producir descendencia.

Este es un operador muy apropiado para el ajuste fino local y también para encauzar correctamente buenas direcciones de búsqueda.

Es importante darse cuenta de que como los PEs numéricos usan una representación natural (sin codificación) se debe comprobar al usar los operadores genéticos que los resultados no se salen de sus correspondientes intervalos $[l_j, u_j]$; en los AGs eso no era necesario porque se hacía implícitamente al diseñar el mecanismo de codificación. Esto introduce el problema del tratamiento de las restricciones. Para problemas con restricciones muy sencillas (como en el caso anterior, en que las restricciones únicamente son de intervalo, ie., $l_j \leq x_j \leq u_j$), resulta muy cómodo hacer la comprobación directamente y eliminar las soluciones no factibles; sin embargo, este procedimento de filtrado va contra la filosofía de los PEs y además resulta inapropiado para tratar restricciones no triviales.

Seguidamente se verá como consideran las restricciones los PEs al resolver problemas numéricos.

5.1.1 Consideración de las restricciones en los Programas de Evolución numéricos

Los PEs tratan de incorporar las restricciones implícitamente dentro de la representación y/o los operadores genéticos. Lo ideal sería que el espacio de búsqueda coincidiera con el dominio del problema y que los operadores genéticos fueran *cerrados*, esto es, proporcionaran siempre resultados factibles. En el caso de los PEs numéricos la representación directa no permite identificar el espacio de búsqueda con el dominio del problema, pero en muchos casos sí que se pueden adaptar los operadores genéticos para que sean cerrados. Ésto es muy importante, pues basta con generar una población inicial factible para garantizar que el algoritmo nunca se saldrá del dominio del problema.

Considérese el típico problema de optimización paramétrica

$$[OP] \left\{ \begin{array}{rl} \text{maximizar} & f(\mathbf{x}) \\ \text{sujeto a} & \mathbf{x} \in \mathcal{X} \subseteq \mathcal{R}^m \end{array} \right.$$

Si se admite que el dominio del problema $\mathcal{X}$ es convexo se verifican, entre otras, estas dos propiedades:

1. Dados dos puntos cualesquiera $\mathbf{x}$ e $\mathbf{y}$ dentro del dominio $\mathcal{X}$, para todo $a \in [0, 1]$ se cumple que
$$a \cdot \mathbf{x} + (1 - a) \cdot \mathbf{y}$$
también está dentro del dominio del problema.

2. Sean $\mathbf{x}$ un punto del dominio y $\wp$ una recta que lo contiene. En tal caso, la recta $\wp$ siempre intersecta a las fronteras de $\mathcal{X}$ en dos puntos.

La primera propiedad es de interés porque garantiza que el cruce aritmético será cerrado, la segunda servirá para adaptar dinámicamente el rango de aplicación de los operadores genéticos con el fin de que en todo momento conserven la factibilidad. La idea es así: sea $\mathbf{x}$ un punto factible y $\wp$ la recta paralela al j-ésimo eje coordenado que pasa por $\mathbf{x}$. La segunda propiedad garantiza que $\wp$ intersecta al dominio $\mathcal{X}$ en dos puntos, de los que sólo son de interés las j-ésimas coordenadas; esas coordenadas se denotan como $l_j(\mathbf{x})$ y $u_j(\mathbf{x})$ conviniéndose en que $l_j \leq u_j$; si la desigualdad es estricta se dice que la j-ésima componente del dominio es *móvil* a la altura de $\mathbf{x}$. Seguidamente se verá cómo hacer uso de esas propiedades al implantar un PE numérico.

5.1.1.1 Operadores genéticos de rango dinámico

Si el dominio $\mathcal{X}$ del problema [OP] es convexo y se dispone de un procedimiento útil con el que calcular los valores l_j y u_j asociados a cada punto factible $\mathbf{x}$

se puede hacer que los operadores genéticos sean cerrados adaptándolos de la siguiente manera:

Mutación numérica estacionaria de rango dinámico: Es igual que la mutación numérica estacionaria vista en el apartado anterior, salvo que ahora las cotas del gen a mutar u_j y l_j son calculadas en cada ocasión.

Mutación numérica no estacionaria de rango dinámico: Lo mismo de antes, cambiando lo que corresponda.

Mutación de fronteras: Se hace que $x_j[t+1]$ valga $u_j(\mathbf{x}[t])$ ó $l_j(\mathbf{x}[t])$ con igual probabilidad. Es conveniente introducir este operador dado que las soluciones de muchos problemas acostumbran a estar en las fronteras.

Cruce aritmético: Este operador no precisa ninguna modificación, ya que la convexitud de las restricciones garantiza la factibilidad de los descendientes, independientemente de lo que valga el parámetro $a \in [0,1]$.

Cruce aritmético sencillo de rango dinámico: Como ya se sabe, consiste en seleccionar un punto de cruce $j \in \{1,\ldots,m\}$ y cruzar aritméticamente sólo las componentes j-ésimas de ambos progenitores, es decir

$$\begin{aligned} x_j[t+1] &= a \cdot x_j[t] + (1-a) \cdot y_j[t] \\ y_j[t+1] &= a \cdot y_j[t] + (1-a) \cdot x_j[t] \end{aligned}$$

donde ahora el parámetro a varía dinámicamente con el fin de que no se pierda la factibilidad.

Se demuestra fácilmente que el cruce aritmético sencillo es cerrado siempre que a esté dentro del siguiente rango:

$$a \in \left\{ \begin{array}{lll} [\max\{\alpha,\beta\}, \min\{\delta,\gamma\}] & \text{si} & x_j > y_j \\ [\max\{\gamma,\delta\}, \min\{\alpha,\beta\}] & \text{si} & x_j < y_j \end{array} \right.$$

$$\text{siendo} \quad \alpha := \frac{l_j(\mathbf{y}) - y_j}{x_j - y_j} \qquad \gamma := \frac{l_j(\mathbf{x}) - x_j}{y_j - x_j}$$
$$\beta := \frac{u_j(\mathbf{x}) - x_j}{y_j - x_j} \qquad \delta := \frac{u_j(\mathbf{y}) - y_j}{x_j - y_j}$$

para cada instante t.
En ausencia de otras indicaciones se suele dar al parámetro a un valor aleatorio dentro de su rango. Evidentemente, si x_j coincide con y_j no es necesario hacer el cruce.

Cruce mixto de rango dinámico: En este caso no es fácil calcular el valor de a que garantiza la factibilidad. En la práctica se le da un valor inicial discrecional y se modifica sólo cuando se pierda la factibilidad. La experiencia muestra que según progresa la búsqueda ésto ocurre con menor frecuencia.

Adaptando los operadores de esa manera y partiendo de una población inicial de puntos factibles queda garantizada la factibilidad de todas las soluciones que se obtengan. Como acaba de verse, para adaptar el rango de aplicación de los operadores genéticos es necesario calcular, a partir del conjunto de restricciones, los valores l_j y u_j asociados a cada punto factible $\mathbf{x} \in \mathcal{X}$. En la práctica eso sólo es viable cuando el dominio del problema queda completamente definido a través de un conjunto de desigualdades lineales del estilo de

$$\mathbf{A} \cdot \mathbf{x} + \mathbf{b} \geq \mathbf{0}$$

si no es ese el caso habrá que recurrir a procedimientos de linealización.

5.1.1.2 Eliminación de las restricciones lineales de igualdad

En muchos problemas de optimización paramétrica se pueden despejar tantas variables como restricciones de igualdad —abreviadamente, *ligaduras*— haya sin más que resolver correspondiente subsistema de ecuaciones y modificar apropiadamente la función objetivo y el resto de restricciones. Es conveniente hacerlo porque de ese modo se reduce el extensión del espacio de búsqueda. Una vez más, ésto sólo es viable en la práctica cuando las restricciones son lineales.

Considérese un problema de optimización paramétrica en el que el dominio $\mathcal{X}$ queda definido mediante un conjunto de restricciones del siguiente modo:

– m restricciones de dominio, ie., del tipo

$$l_j \leq x_j \leq u_j \qquad (\forall j = 1, \ldots, m)$$

Comúnmente eso se abrevia así

$$\mathbf{l} \leq \mathbf{x} \leq \mathbf{u}$$

donde, naturalmente, $\mathbf{l} := (l_1, \ldots, l_m)$ y $\mathbf{u} := (u_1, \ldots, u_m)$ son los vectores (especificados) de cotas y $\mathbf{x} := (x_1, \ldots, x_m)$ es un vector de incógnitas.

– p ligaduras lineales de igualdad, ie., del tipo

$$\mathbf{A} \cdot \mathbf{x} = \mathbf{b}$$

siendo $\mathbf{x}$ el de antes, y $\mathbf{b} := (b_1, \ldots, b_p)$ y $\mathbf{A} := [a_{ij}] \in \mathcal{R}^{p \times m}$ especificados.

– q restricciones lineales de desigualdad, ie., del tipo

$$\mathbf{C} \cdot \mathbf{x} \leq \mathbf{d}$$

siendo $\mathbf{x}$ el de antes, y $\mathbf{d} := (d_1, \ldots, d_q)$ y $\mathbf{C} := [c_{ij}] \in \mathcal{R}^{q \times m}$ especificados.

Se pueden eliminar las p restricciones de igualdad reorganizando $\mathbf{x}$ en p variables independientes —en la jerga del ramo *variables básicas*— y otras $m-p$ variables dependientes (de las anteriores), así:

$$\mathbf{x} := [\mathbf{x_B}|\mathbf{x_{NB}}]$$

Esa reorganización implica la reordenación de las columnas de $\mathbf{A}$ y $\mathbf{C}$ y de los elementos de $\mathbf{l}, \mathbf{u}, \mathbf{b}$ y $\mathbf{d}$. De esta manera, la igualdad matricial $\mathbf{A} \cdot \mathbf{x} = \mathbf{b}$ se reescribe así

$$[\mathbf{A_B}|\mathbf{A_{NB}}] \cdot \left[\frac{\mathbf{x_B}}{\mathbf{x_{NB}}} \right] = \mathbf{b} \quad \rightarrow \quad \mathbf{A_B} \cdot \mathbf{x_B} + \mathbf{A_{NB}} \cdot \mathbf{x_{NB}} = \mathbf{b}$$

donde $\mathbf{A}_B \in \mathcal{R}^{p \times p}$ debe ser inversible, para así poder eliminar las p variables básicas y reemplazarlas por una combinación lineal de las restantes de este modo

$$\mathbf{x_B} = \mathbf{A_B^{-1}} \cdot \mathbf{b} - \mathbf{A_B^{-1}} \cdot \mathbf{A_{NB}} \cdot \mathbf{x_{NB}}$$

Es estas circunstancias las p restricciones de igualdad son superfluas, y las otras toman la siguiente forma:

- Las m restricciones de dominio se transforman en estos dos grupos:

$$\mathbf{l_B} \leq \mathbf{A_B^{-1}} \cdot \mathbf{b} - \mathbf{A_B^{-1}} \cdot \mathbf{A_{NB}} \cdot \mathbf{x_{NB}} \leq \mathbf{u_B}$$

$$\mathbf{l_{NB}} \leq \mathbf{x_{NB}} \leq \mathbf{u_{NB}}$$

- Las q restricciones lineales de desigualdad se transforman así:

$$\begin{array}{rcllcl} \mathbf{C} \cdot \mathbf{x} & \leq & \mathbf{d} & \rightarrow & \mathbf{C_B} \cdot \mathbf{x_B} + \mathbf{C_{NB}} \cdot \mathbf{x_{NB}} & \leq \ \mathbf{d} \\ & & & \rightarrow & \mathbf{C_B} \left(\mathbf{A_B^{-1}} \cdot \mathbf{b} - \mathbf{A_B^{-1}} \cdot \mathbf{A_{NB}} \cdot \mathbf{x_{NB}} \right) + \mathbf{C_{NB}} \cdot \mathbf{x_{NB}} & \geq \ \mathbf{d} \end{array}$$

de manera que

$$\left(\mathbf{C_{NB}} - \mathbf{C_B} \cdot \mathbf{A_B^{-1}} \cdot \mathbf{A_{NB}} \right) \cdot \mathbf{x_{NB}} \geq \mathbf{d} - \mathbf{C_B} \cdot \mathbf{A_B^{-1}} \cdot \mathbf{b}$$

Por último, se deben sustituir en la función objetivo $f(\mathbf{x_B}, \mathbf{x_{NB}})$ las variables despejadas $\mathbf{x_B}$ por sus respectivos valores.

Ejemplo: Supóngase que se desea resolver mediante un PE numérico el problema de transporte entre dos fuentes y tres destinos que se muestra en la siguiente tabla:

	$d_1 = 3$	$d_2 = 4$	$d_3 = 8$
$s_1 = 5$	x_{11}	x_{12}	x_{13}
$s_2 = 10$	x_{21}	x_{22}	x_{23}

En ella las variables x_{ij} indican la cantidad de s_i que se manda a d_j. De lo que se trata es de hallar los valores de $\mathbf{x} := (x_{11}, x_{12}, x_{13}, x_{21}, x_{22}, x_{23})$ que hacen mínima cierta función $f(\mathbf{x})$ —no necesariamente lineal— con la que se cuantifican los costes totales de transporte.

Independientemente de cual sea la solución óptima, las x_{ij} están ligadas por las siguientes ecuaciones:

$$\begin{aligned} x_{11} + x_{12} + x_{13} &= 5 \\ x_{21} + x_{22} + x_{23} &= 10 \\ x_{11} + x_{21} &= 3 \\ x_{12} + x_{22} &= 4 \\ x_{13} + x_{23} &= 8 \end{aligned}$$

Evidentemente, una de ellas es redundante (vg., la última), por lo que se puede suprimir.

Por otra parte, dada la naturaleza del problema, no se permite ninguna x_{ij} negativa, lo que a su vez obliga a que $x_{ij} \leq \min\{s_i, d_j\}$. En definitiva, esto hace que los vectores de cotas $\mathbf{l}$ y $\mathbf{u}$ valgan respectivamente:

$$\mathbf{l} = (0, 0, 0, 0, 0, 0) \qquad \mathbf{u} = (3, 4, 5, 3, 4, 8)$$

En principio, para resolver ese problema deberían usarse individuos con forma de vectores reales de tamaño 6; sin embargo, es posible hacer la resolución con vectores de tamaño 2 eliminando cuatro variables de la siguiente manera.

Las cinco ecuaciones de ligadura se pueden expresar matricialmente así:

$$\left[\begin{array}{cccc|cc} 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \end{array}\right] \cdot \left[\begin{array}{c} x_{11} \\ x_{12} \\ x_{13} \\ x_{21} \\ \hline x_{22} \\ x_{23} \end{array}\right] = \left[\begin{array}{c} 5 \\ 10 \\ 3 \\ 4 \end{array}\right]$$

Dado que las cuatro primeras columnas son independientes se pueden tomar sus respectivas variables como variables básicas y las otras dos como variables no básicas. Así se tiene que:

$$\mathbf{A_B} = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{bmatrix} \quad \mathbf{A_{NB}} = \begin{bmatrix} 0 & 0 \\ 1 & 1 \\ 0 & 0 \\ 1 & 0 \end{bmatrix} \quad \mathbf{x_B} = \begin{bmatrix} x_{11} \\ x_{12} \\ x_{13} \\ x_{21} \end{bmatrix} \quad \mathbf{x_{NB}} = \begin{bmatrix} x_{22} \\ x_{23} \end{bmatrix}$$

Despejando de la ecuación

$$[\mathbf{A_B} | \mathbf{A_{NB}}] \cdot \left[\begin{array}{c} \mathbf{x_B} \\ \hline \mathbf{x_{NB}} \end{array}\right] = \mathbf{b}$$

se obtendrá

$$\mathbf{x_B} = \begin{bmatrix} x_{11} \\ x_{12} \\ x_{13} \\ x_{21} \end{bmatrix} = \begin{bmatrix} -7 \\ 4 \\ 8 \\ 10 \end{bmatrix} - \begin{bmatrix} -1 & -1 \\ 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_{22} \\ x_{23} \end{bmatrix}$$

quedando así expresadas las cuatro variables básicas x_{11}, x_{12}, x_{13} y x_{21} en función de las no básicas, x_{22} y x_{23}. De esta manera las restricciones de dominio $\mathbf{l} \leq \mathbf{x} \leq \mathbf{u}$ se transforman en:

$$\begin{bmatrix} 7 \\ -4 \\ -8 \\ -10 \end{bmatrix} \leq \begin{bmatrix} 1 & 1 \\ -1 & 0 \\ 0 & -1 \\ -1 & -1 \end{bmatrix} \cdot \begin{bmatrix} x_{22} \\ x_{23} \end{bmatrix} \leq \begin{bmatrix} 10 \\ 0 \\ -3 \\ -7 \end{bmatrix} \qquad \text{y} \qquad \begin{bmatrix} 0 \\ 0 \end{bmatrix} \leq \begin{bmatrix} x_{22} \\ x_{23} \end{bmatrix} \leq \begin{bmatrix} 4 \\ 8 \end{bmatrix}$$

o, lo que es equivalente:

$$7 \leq x_{22} + x_{23} \leq 10$$

$$0 \leq x_{22} \leq 4$$

$$3 \leq x_{23} \leq 8$$

Como no hay restricciones de desigualdad del tipo $\mathbf{C} \cdot \mathbf{x} \leq \mathbf{d}$, solamente queda por transformar la función objetivo $f(\mathbf{x})$ en

$$g(\mathbf{x_{NB}}) := f(\mathbf{x_B}, \mathbf{x_{NB}}) = f(-7 + x_{23} + x_{33}, 4 - x_{23}, 8 - x_{33}, 10 - x_{23} - x_{33}, x_{23}, x_{33})$$

para que sólo dependa de las variables no básicas. Así se ha reducido por completo el problema original a uno equivalente de tamaño 2.

Con ese nuevo conjunto de restricciones se puede ilustrar muy fácilmente cómo se realiza el cálculo de los vectores $\mathbf{l}(x_{22}, x_{23})$ y $\mathbf{u}(x_{22}, x_{23})$. Sean, por ejemplo, $x_{22} = 2$ y $x_{23} = 6$. Para hallar la primera coordenada de $\mathbf{l}$ y $\mathbf{u}$ se sustituye $x_{22} = 2$ en las desigualdades anteriores y se despeja x_{23}, para la segunda coordenada se hace lo recíproco. Operando es muy sencillo ver que se obtiene en cada caso

$$1 \leq x_{22} \leq 4 \qquad \text{y} \qquad 5 \leq x_{23} \leq 8$$

de manera que

$$\mathbf{l}(x_{22}, x_{23}) = (1, 5) \qquad \text{y} \qquad \mathbf{u}(x_{22}, x_{23}) = (4, 8)$$

sobre el punto especificado.

5.1.2 Programas de Evolución numéricos con restricciones no lineales

No siempre es posible definir el dominio del problema [OP] a través de un conjunto de restricciones lineales; en tal caso el procedimiento anterior no será aplicable, y se deberán tratar aparte las restricciones no lineales. Antes de hacerlo conviene repasar someramente cómo se ha resuelto tradicionalmente este problema.

Se han propuesto muchos métodos para resolver el problema general de programación estática

$$[\mathrm{PE}] \begin{cases} \text{maximizar} & f(\mathbf{x}) \\ \text{sujeto a} & \mathbf{g}(\mathbf{x}) \geq \mathbf{0} \quad \text{y} \quad \mathbf{x} \geq \mathbf{0} \end{cases}$$

La mayoría de ellos tratan de transformar ese problema en una sucesión de problemas cuadráticos resolubles que converge a la verdadera solución. El problema está en que para construir esa sucesión es necesario calcular —de modo explícito o implícito— la matriz de segundas derivadas de la función objetivo (matriz hessiana), la cual presenta graves problemas de acondicionamiento en las proximidades del óptimo. Se trata, pues, de usar los PEs para evitar el cálculo de esa matriz.

El modo más sencillo de introducir los PEs en el problema es utilizando el *método de la función de penalización cuadrática secuencial.* En él se sustituye el problema [PE] por otro irrestricto

$$[\mathrm{PEI}] \qquad \text{maximizar } \phi(\mathbf{x}, K) := f(\mathbf{x}) + \frac{1}{2K}\,\mathbf{a}'(\mathbf{x}) \cdot \mathbf{a}(\mathbf{x})$$

donde K es un número positivo y $\mathbf{a}(\mathbf{x})$ es un vector de restricciones bloqueadas en el punto $\mathbf{x}$.
Si se hace uso del teorema de Fiacco-McCormick (ver [MICH94,p158] y un enunciado más riguroso en [BAZA93]) resulta que las soluciones de [PE] y [PEI] se aproximan hasta hacerse iguales en el límite cuando $K \to 0$, por lo que se podrá hallar el óptimo resolviendo el problema [PEI] para una sucesión positiva de valores descendientes de K. En este caso la matriz hessiana aparecería al resolver analíticamente los problemas [PEI]; sin embargo, existe la posibilidad de evitar tal cálculo resolviendo esa sucesión de problemas a través de PEs numéricos.

Broyden y Attia propusieron implantar la anterior estrategia de la siguiente manera: se separan las restricciones lineales de las no lineales y se resuelven sucesivamente los problemas [PEI] sujetos a las restricciones lineales utilizando el método descrito en el apartado anterior. Antes de plasmarlo en un seudocódigo conviene hacer las siguientes definiciones:

Siendo G el conjunto de restricciones con el que se delimita completamente el dominio $\mathcal{X}$ se definen los siguientes subconjuntos ($\subseteq G$):

L	es el subconjunto de restricciones lineales
Ne	es el subconjunto de restricciones no lineales de igualdad
Ni	es el subconjunto de restricciones no lineales de desigualdad
$V(\mathbf{x})$	es el subconjunto ($\subseteq Ni$) de restricciones violadas para el punto $\mathbf{x}$
$S(\mathbf{x})$	es el subconjunto ($\subseteq Ni$) de restricciones satisfechas para el punto $\mathbf{x}$
$A(\mathbf{x})$	es el subconjunto de restricciones bloqueadas para el punto $\mathbf{x}$. Se calcula como $A(\mathbf{x}) = V(\mathbf{x}) \cup Ne$

A efectos prácticos, se considera que una restricción $g_j \in Ni$ se ha violado en el punto $\mathbf{x}$ cuando $g_j(\mathbf{x}) < \epsilon$ siendo ϵ un parámetro del método.

Haciendo uso de esas definiciones la estructura genérica del algoritmo queda así:

```
BeginAlgorithm{NonLinearOptimization}
 Xcur = SelectStartingPoint();            No necesariamente factible
 M = InitPenalty();                       M := 1/2K
 [L,Ne,Ni] = SplitConstraints(G);
 for( t = 0; t < MaxIter; t++ ) {
  [Sx,Vx] = SplitIneqConstr(Ni,Xcur);
  Ax = Join(Ne,Vx);
  Xopt = Solve( f(x)+M*Trn(Ax)*Ax, L, Xcur );
  if( ChkTermCond(Xopt,Fopt) )
   return;
  Xcur = Xopt;
  M = Incr(M,t);                          Gralmte., M = Kincr × M
 }
EndAlgorithm
```

De esa manera se puede calcular la dirección de búsqueda evitando los cálculos implícitos o explícitos de gradientes o hessianas.

Nótese que el PE interior arranca la primera vez con *PopSize* copias idénticas del punto inicial [1] y las restantes veces con otras tantas copias del mejor punto obtenido hasta entonces. Para conseguir variedad en la población el operador mutación se deberá usar frecuentemente las primeras etapas de la evolución.

Zbigniew Michalewicz ha implantado detalladamente ese esquema en una aplicación llamada GENOCOP (acrónimo para *Genetic Algorithm for Numerical Optimization of Constrained Problems*, esto es, algoritmo genético para optimización numérica de problemas con restricciones). Los archivos fuente son de dominio público y se pueden obtener a través de un "ftp anónimo" a la dirección `ftp://ftp.uncc.edu/coe/evol/genocop3.tar.Z`. La aplicación fue programada en lenguaje ANSI-C, por lo que se puede compilar sin problemas en gran

[1] Es una ventaja sobre otros métodos el hecho de que el punto inicial no deba ser necesariamente factible, pues con restricciones no lineales no es fácil obtener una solución inicial factible.

cantidad de plataformas. Además se incluyen muchos ejemplos de problemas difíciles de programación estática, todos los cuales son resueltos con éxito. La documentación no es exhaustiva, por lo que se recomienda consultar el capítulo 7 de [MICH94], donde se describe detalladamente su funcionamiento.

5.1.3 Mejoras a los Programas de Evolución numéricos

La representación natural usada por los PEs numéricos permite introducir nuevos procedimientos generales de mejora, la mayoría de los cuales están inspirados en otros métodos de optimización numérica. Se estudiarán aquí los dos más usados.

5.1.3.1 Codificación dinámica de los parámetros

Se trata de ajustar dinámicamente la precisión de las soluciones codificándolas de la siguiente manera. Cada variable contiene un número en coma flotante de tamaño especificado; cuando el PE converge, se salva el dígito más significativo de la mantisa y se desplazan los demás hacia la izquierda, introduciendo uno nuevo. Ese nuevo dígito en la posición menos significativa incrementa la precisión de la búsqueda. El proceso se repite hasta cumplir la condición de terminación. También es recomendable guardar el exponente y aprovechar los espacios que ha dejado libres para incrementar más aún la precisión.

5.1.3.2 PEs con búsqueda por perturbación local

La búsqueda por perturbación local es un procedimiento genérico utilizado por muchos métodos de optimización que consiste en hallar nuevos candidatos a solución perturbando sucesivamente los ya existentes. Aplicado a los PEs, la idea consiste en realizar la búsqueda genética en dos niveles distintos:

1 Nivel $\mathbf{v}$, o de las soluciones potenciales al problema
2 Nivel $\delta\mathbf{v}$, o nivel incremental

La mejor solución encontrada en el nivel $\mathbf{v}$ por una sóla aplicación de un PE se guarda y se toma como referencia para el nivel $\delta\mathbf{v}$. A continuación se ejecutan varias iteraciones del nivel incremental hasta que haya convergencia; así se ha encontrado la mejor modificación $\delta\mathbf{v}$ para la solución $\mathbf{v}$. Con ese resultado se actualiza la solución $\mathbf{v}$ y se repite el proceso. Evidentemente, para evaluar en el nivel incremental el individuo $\delta\mathbf{v}$ se considera el perturbado $\mathbf{v} + \delta\mathbf{v}$. A la larga, lo que se está haciendo es separar las etapas de aproximación global y de ajuste fino.
En forma esquemática,

```
BeginProcedure{DeltaCoding}
BestV = ApplyEP_V;
```

```
while (not TERM_COND) {
   BestDeltaV = ApplyEP_DeltaV;
   BestV = BestV + BestDeltaV;
}
EndProcedure
```

Nótese que en cada aplicación del bucle interior se reinicializa la población de perturbaciones $\delta\mathbf{v}$

Este mecanismo se puede introducir también en PEs no numéricos, sin más que dar el significado de 'solución perturbada' a $\mathbf{v} + \delta\mathbf{v}$.

5.1.4 Mejoras específicas

Aparte de estos procedimientos genéricos de mejora, se están desarrollando gran cantidad de procedimientos específicos orientados hacia la representación y los operadores genéticos. En efecto, en la presente sección se ha descrito con cierto detalle cómo se resuelve un problema general de optimización numérica utilizando PEs; sin embargo, se puede mejorar el funcionamiento del PE numérico al resolver un problema concreto de optimización proporcionándole operadores específicos e incluso una representación distinta a la vectorial. Si se hace la elección con fundamento, ello no debería implicar una gran cantidad de conocimiento específico adicional (esto es, de esfuerzo de implantación adicional).

En los capítulos 7 y 9 de [MICH94] se describen dos PEs diseñados para resolver sendos problemas específicos de optimización numérica. Uno de ellos es el clásico problema del transporte de la Investigación de Operaciones (ver [HILL82]) en el que la función objetivo no es necesariamente lineal; se usa una representación matricial y operadores específicos de cruce y mutación. El otro es el problema del control óptimo con función de costes arbitraria y condiciones frontera genéricas. Ambos problemas podrían ser resueltos por el método general aquí descrito —e implantado en la aplicación GENOCOP—, pero las prestaciones son notablemente inferiores. Una vez más se vuelve al compromiso entre generalidad y eficiencia ya discutido en 1.2.3.

5.2 Programas de Evolución especializados en optimización combinatoria

En la optimización combinatoria no existe un *problema básico* análogo al problema de programación estática de la optimización numérica. Los problemas de optimización combinatoria son extremadamente variados y ninguno se puede situar en la posición central. Ello no obsta para que existan problemas más característicos o más fecundos que otros. En particular, ciñéndose al subconjunto de los problemas basados en el orden, existe un problema de ese tipo que destaca sobre los demás: es el problema del viajante de comercio, o abreviadamente PVC (en la bibliografía inglesa, *Travelling Salesperson Problem* o TSP). Más allá de su valor teórico —por ser un problema NP-completo— o de su valor práctico —vg., en el trazado de circuitos VLSI— el PVC es del máximo interés por ser un ejemplo arquetípico de problema basado en el orden, pudiéndose afirmar que una buena técnica para resolver el PVC serviría también para resolver otros muchos problemas basados en el orden y, en general, otros problemas combinatorios. En lo que respecta a los PEs, esta preeminencia se acentúa por el hecho de que se han propuesto para resolverlo[2] múltiples representaciones, cada una con sus operadores asociados; ninguna de ellas destaca especialmente sobre las demás, pero muchas son lo suficientemente generales como para poder ser útiles en otros problemas basados en el orden. Por todos estos motivos, se estudiará con detenimiento.

El ***problema del viajante de comercio*** se enuncia así:

> Dadas m ciudades (o vértices) $\{1, 2, \ldots, m\}$ y los costes de viajar de unas a otras $c_{ij} \quad (i, j \in \{1, \ldots, \mathrm{m}\})$ se trata de calcular un tramo (ie., conjunto de rutas o aristas) completo (ie., que visite todas las ciudades), cerrado (que comience y termine en la misma ciudad) y conexo que haga mínimo el coste total del recorrido.
> Se admite por defecto la simetría de la matriz de costes, de manera que éstos son asimilables a unas 'distancias generalizadas'.

En la figura 5.3 se muestra un candidato a solución para un PVC con siete ciudades.

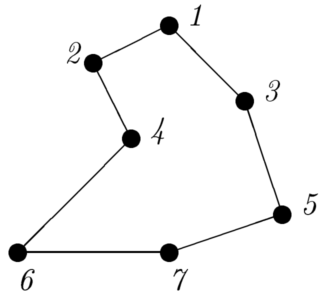

Figura 5.3: Problema del viajante de comercio

Se ve claramente que el espacio de búsqueda para el PVC es el de las permutaciones de las m ciudades o vértices. Cualquier permutación representa a un

[2] Esto es, para calcular cuasi-óptimos aceptables, que es lo que hacen los algoritmos evolutivos.

tramo completo, cerrado y conexo (abreviadamente, *recorrido*) y por tanto a un candidato a solución.

Es un hecho que no existe un modo práctico de codificar el conjunto de las permutaciones de m elementos mediante cadenas binarias de bits de modo tal que los operadores genéticos clásicos sean eficientes en la resolución del PVC. A estas alturas parece claro que la representación binaria es demasiado pobre para este problema y que la representación mediante cadenas de números enteros es la que mejor combina la sencillez y la eficiencia; por eso es la que se usará en esta sección por defecto.

Sea cual sea la codificación que se use, resulta relativamente fácil encontrar operadores de mutación (ie., que introduzcan un ligero cambio en un recorrido) cerrados. No ocurre lo mismo con los operadores de cruce, por lo que se deberán estudiar detenidamente para cada representación.

5.2.1 Codificación y operadores de recorrido

La manera más natural de codificar las soluciones del PVC consiste en enumerar las ciudades por orden de recorrido —el sentido es arbitrario, dada la simetría. Por ejemplo, para un PVC con nueve ciudades el recorrido

$$\mathit{5} \to \mathit{1} \to \mathit{7} \to \mathit{8} \to \mathit{9} \to \mathit{4} \to \mathit{6} \to \mathit{2} \to \mathit{3} \hookleftarrow$$

se representará como el individuo $\langle \mathtt{5,1,7,8,9,4,6,2,3} \rangle$. Nótese que el individuo $\langle \mathtt{8,9,4,6,2,3,5,1,7} \rangle$ también representa al mismo recorrido, sólo que comenzando por la ciudad *8*; lo mismo ocurre con el individuo $\langle \mathtt{3,2,6,4,9,8,7,1,5} \rangle$, que sólo se distingue en el sentido del recorrido.

El problema de esta codificación radica en que el cruce externo clásico (entre genes) no es cerrado; de hecho, casi nunca proporciona individuos válidos. Por este motivo se debe recurrir a otros operadores (cerrados) de cruce. De entre ellos destacan los siguientes:

Cruce por emparejamiento parcial (PMX): Consiste en elegir un tramo de uno de los progenitores y cruzar preservando el orden y la posición de la mayor cantidad posible de ciudades del otro. Se entenderá mejor sobre un ejemplo: siendo

$$\begin{aligned} \mathbf{v} &= \langle \mathtt{1,2,3,4,5,6,7,8,9} \rangle \\ \mathbf{w} &= \langle \mathtt{4,5,2,1,8,7,6,9,3} \rangle \end{aligned}$$

se desea hallar la descendencia a través del cruce PMX entre los genes $3^{\underline{o}}$ y $7^{\underline{o}}$. El modo de operación se resume en estos cuatro pasos:

1. Se selecciona un recorrido parcial eligiendo al azar dos puntos de corte. En el presente caso, ya están dados:

$$\langle 1,2,3 \mid 4,5,6,7 \mid 8,9\rangle$$
$$\langle 4,5,2 \mid 1,8,7,6 \mid 9,3\rangle$$

2. Se intercambian los segmentos situados entre los puntos de corte:

$$\langle \mathtt{x},\mathtt{x},\mathtt{x} \mid 1,8,7,6 \mid \mathtt{x},\mathtt{x}\rangle$$
$$\langle \mathtt{x},\mathtt{x},\mathtt{x} \mid 4,5,6,7 \mid \mathtt{x},\mathtt{x}\rangle$$

   Los símbolos `x` se interpretan como 'no especificado'.
   Ese intercambio define también un conjunto de emparejamientos

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

   que serán empleados para despejar las `x`.

3. Se especifican las `x` de los progenitores originales que no planteen conflicto, es decir:

$$\langle \mathtt{x},2,3 \mid 1,8,7,6 \mid \mathtt{x},9\rangle$$
$$\langle \mathtt{x},\mathtt{x},2 \mid 4,5,6,7 \mid 9,3\rangle$$

4. Las `x` que plantean conflicto se reemplazan por su pareja. Por ejemplo, el primer elemento del primer progenitor está repetido, por lo que se sustituye por 4:

$$\langle 4,2,3 \mid 1,8,7,6 \mid 5,9\rangle$$
$$\langle 1,8,2 \mid 4,5,6,7 \mid 9,3\rangle$$

De esta manera siempre se obtienen soluciones factibles.

Cruces por orden (OX): El cruce por orden básico consiste en elegir para cada descendiente un tramo de uno de los progenitores y a la vez preservar el orden relativo de todas las ciudades del otro. Para el ejemplo anterior el modo de operación es éste:

1. Este paso es el mismo de antes, ie., se selecciona un recorrido parcial eligiendo al azar dos puntos de corte:

$$\langle 1,2,3 \mid 4,5,6,7 \mid 8,9\rangle$$
$$\langle 4,5,2 \mid 1,8,7,6 \mid 9,3\rangle$$

2. Es el mismo de antes, pero sin prestar atención a los emparejamientos:

$$\langle \mathtt{x},\mathtt{x},\mathtt{x} \mid 4,5,6,7 \mid \mathtt{x},\mathtt{x}\rangle$$
$$\langle \mathtt{x},\mathtt{x},\mathtt{x} \mid 1,8,7,6 \mid \mathtt{x},\mathtt{x}\rangle$$

3. Para cada progenitor se parte de uno de los puntos de corte y se copian las ciudades del otro progenitor conservando el orden relativo y omitiendo las que ya estén presentes; al llegar al final de la cadena se continúa por el principio hasta retornar al punto de partida.
En el ejemplo presente, se parte del segundo punto de corte para ambos progenitores; así resulta que para el primero se obtiene el siguiente descendiente:

$$\langle \mathtt{2}, \mathtt{1}, \mathtt{8} \mid \mathtt{4}, \mathtt{5}, \mathtt{6}, \mathtt{7} \mid \mathtt{9}, \mathtt{3} \rangle$$

Obsérvese que el primer elemento no es ni `4` ni `5`, porque ya están incluidos ambos. Análogamente, para el segundo progenitor se obtiene este otro descendiente:

$$\langle \mathtt{3}, \mathtt{4}, \mathtt{5} \mid \mathtt{1}, \mathtt{8}, \mathtt{7}, \mathtt{6} \mid \mathtt{9}, \mathtt{2} \rangle$$

Se hace notar que el cruce por orden sólo considera el orden relativo de las ciudades, no su posición. No obstante, existen dos variantes de este cruce que sí que toman en cuenta las posiciones. Ambas son profusamente utilizadas en problemas de planificación de tareas:

Cruce por orden con posiciones prioritarias: La única diferencia con el cruce anterior estriba en que no se elige un tramo para intercambiarlo entre los progenitores, sino un conjunto de posiciones al azar. El resto es igual.

Cruce por orden con orden prioritario: Los individuos no intercambian ciudades, sino el orden relativo existente entre ellas. Supóngase que para los individuos del ejemplo anterior se eligen para el intercambio de orden las posiciones 3ª, 4ª, 6ª y 9ª. En el primer progenitor el orden de las ciudades en esas posiciones es

$$\mathit{3} \to \mathit{4} \to \mathit{6} \to \mathit{9}$$

En el segundo progenitor esas ciudades ocupan las posiciones 9ª, 1ª, 7ª y 8ª, por lo que el primer descendiente será una copia del segundo progenitor en todas las posiciones salvo en ésas, es decir

$$\langle \mathtt{x}, \mathtt{5}, \mathtt{2}, \mathtt{1}, \mathtt{8}, \mathtt{7}, \mathtt{x}, \mathtt{x}, \mathtt{x} \rangle$$

En ellas se colocarán las ciudades seleccionadas conservando su orden relativo, o sea

$$\langle \mathtt{3}, \mathtt{5}, \mathtt{2}, \mathtt{1}, \mathtt{8}, \mathtt{7}, \mathtt{4}, \mathtt{6}, \mathtt{9} \rangle$$

y ese será el primer descendiente. El segundo se obtendrá repitiendo el mismo procedimiento para el segundo progenitor:

$$\langle \mathtt{2}, \mathtt{1}, \mathtt{7}, \mathtt{4}, \mathtt{5}, \mathtt{6}, \mathtt{3}, \mathtt{8}, \mathtt{9} \rangle$$

Cruce por ciclos (CX): Se trata de que cada ciudad herede sucesivamente la posición de uno de los progenitores. Resulta de gran ayuda construir la tabla de posiciones; para el presente ejemplo esa tabla es

	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
v	1º	2º	3º	4º	5º	6º	7º	8º	9º
w	4º	3º	9º	1º	2º	7º	6º	5º	8º

y se opera completando 'ciclos de sucesión' así:

1. Para el primer descendiente se parte de la primera ciudad del primer progenitor

$$\langle \mathtt{1}, \mathtt{x}, \mathtt{x}, \mathtt{x}, \mathtt{x}, \mathtt{x}, \mathtt{x}, \mathtt{x}, \mathtt{x} \rangle$$

Eso obliga a darle a la ciudad *4* en 4º puesto de ese descendiente

$$\langle \mathtt{1}, \mathtt{x}, \mathtt{x}, \mathtt{4}, \mathtt{x}, \mathtt{x}, \mathtt{x}, \mathtt{x}, \mathtt{x} \rangle$$

quedando así completa el primer ciclo de sucesión.

2. Se toma la segunda ciudad de uno de los progenitores al azar, por ejemplo el primero otra vez. Ello obliga a dar a la ciudad *5* el 5º puesto, y eso a su vez obliga a dar a *8* el 8º puesto, a *9* el 9º puesto y a *3* el 3º, lo que completa el segundo ciclo de sucesión y proporciona

$$\langle \mathtt{1}, \mathtt{2}, \mathtt{3}, \mathtt{4}, \mathtt{5}, \mathtt{x}, \mathtt{x}, \mathtt{8}, \mathtt{9} \rangle$$

3. Se recorre el último ciclo de la siguiente manera: ahora se parte desde la 6ª ciudad del segundo progenitor para obtener por fin el primer descendiente:

$$\langle \mathtt{1}, \mathtt{2}, \mathtt{3}, \mathtt{4}, \mathtt{5}, \mathtt{7}, \mathtt{6}, \mathtt{8}, \mathtt{9} \rangle$$

4. El segundo descendiente se obtiene inmediatamente por complementariedad con el primero:

$$\langle \mathtt{4}, \mathtt{5}, \mathtt{2}, \mathtt{1}, \mathtt{8}, \mathtt{6}, \mathtt{7}, \mathtt{9}, \mathtt{3} \rangle$$

Nótese que en el proceso anterior sólo ha habido tres posibles elecciones; una vez hechas, todas las posiciones quedan determinadas.

Cruce por recombinación de rutas (ERX): Ahora la idea es distinta, pues consiste en construir la descendencia combinando no ya las ciudades de los progenitores sino las rutas que las interconectan. Tomando como base el ejemplo anterior el modo de operación es éste:

1. Se construye la tabla de conectividades entre ciudades de uno u otro progenitor, en el presente caso:

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

2. Se construye el primer descendiente tomando la ciudad inicial de uno de los progenitores, vg., la del segundo, *4*.

3. De entre todas las ciudades a que está conectada la anterior (*5* y *3*) se toma la menos conectada y en caso de empate se toma una de ellas al azar (vg., la *5*). Ésto incrementa la probabilidad de completar un recorrido con éxito.

4. Se repite el paso anterior considerando todas las nuevas ciudades conectadas a la última (es decir, *6* y *2*).

5. Se reitera el paso anterior hasta terminar con éxito o en un "callejón sin salida", en cuyo caso (no muy habitual) se comienza de nuevo. En este caso se termina con éxito, resultando como primer descendiente

$$\langle 4, 5, 6, 7, 8, 1, 2, 3, 9 \rangle$$

6. Para obtener el segundo descendiente se repite el proceso comenzando por la otra ciudad inicial. Ahora existe la posibilidad de llegar a un callejón sin salida:

$$\langle 1, 8, 7, 6, 5, 4, 3, 2, \mathtt{x} \rangle$$

   en cuyo caso se comienza de nuevo hasta lograr un individuo factible, como por ejemplo:

$$\langle 1, 2, 5, 4, 3, 9, 8, 7, 6 \rangle$$

Este operador proporciona excelentes resultados en muchos problemas de optimización combinatoria, motivo por el que se han desarrollado gran cantidad de variantes más o menos específicas; la mayoría de ellas consisten en ampliar el criterio de prioridades, vg., marcando algunas ciudades como prioritarias o como secundarias.

Para terminar, es interesante notar que el operador de inversión sí que es cerrado bajo esta codificación, de hecho parece dar buenos resultados (ver [MICH94,p212]) como operador suplementario.

5.2.2 Codificación ordinal

A la vista de la gran cantidad de operadores vistos para la codificación anterior cabe plantearse si existe una codifición del PVC para la que el cruce clásico sea un operador cerrado y no haga falta, por tanto, recurrir a otros operadores.

Tal codificación existe, se llama *codificación ordinal* y consiste en lo siguiente: previamente se ordenan todas las ciudades en una lista dinámica de referencia según cierto criterio; después, para construir un individuo se van sacando una a una las ciudades recorridas, codificando en el j-ésimo gen del individuo la posición que tiene la j-ésima ciudad en la lista dinámica; ese número necesariamente será un entero entre 1 y $m - j + 1$.

Sea, por ejemplo, la lista dinámica $L = \{\mathit{1}, \mathit{2}, \mathit{3}, \mathit{4}, \mathit{5}, \mathit{6}, \mathit{7}, \mathit{8}, \mathit{9}\}$. Basándose en ella, el recorrido

$$\mathit{4} \to \mathit{5} \to \mathit{2} \to \mathit{1} \to \mathit{8} \to \mathit{7} \to \mathit{6} \to \mathit{9} \to \mathit{3} \hookleftarrow$$

se representa así

$$\langle \mathtt{4}, \mathtt{4}, \mathtt{2}, \mathtt{1}, \mathtt{4}, \mathtt{3}, \mathtt{2}, \mathtt{2}, \mathtt{1} \rangle$$

y, naturalmente, el recorrido

$$\mathit{1} \to \mathit{2} \to \mathit{3} \to \mathit{4} \to \mathit{5} \to \mathit{6} \to \mathit{7} \to \mathit{8} \to \mathit{9} \hookleftarrow$$

se representa así

$$\langle \mathtt{1}, \mathtt{1}, \mathtt{1}, \mathtt{1}, \mathtt{1}, \mathtt{1}, \mathtt{1}, \mathtt{1}, \mathtt{1} \rangle$$

Si se aplica el cruce clásico en el cuarto gen, por ejmeplo, se obtienen los individuos

$$\langle \mathtt{4}, \mathtt{4}, \mathtt{2}, \mathtt{1} \mid \mathtt{1}, \mathtt{1}, \mathtt{1}, \mathtt{1}, \mathtt{1} \rangle \qquad \text{y} \qquad \langle \mathtt{1}, \mathtt{1}, \mathtt{1}, \mathtt{1} \mid \mathtt{4}, \mathtt{3}, \mathtt{2}, \mathtt{2}, \mathtt{1} \rangle$$

que corresponden a los recorridos

$$\mathit{4} \to \mathit{5} \to \mathit{2} \to \mathit{1} \to \mathit{3} \to \mathit{6} \to \mathit{7} \to \mathit{8} \to \mathit{9} \hookleftarrow$$

y

$$\mathit{1} \to \mathit{2} \to \mathit{3} \to \mathit{4} \to \mathit{8} \to \mathit{7} \to \mathit{6} \to \mathit{9} \to \mathit{5} \hookleftarrow$$

ambos factibles, como ya se anticipó.

Pese a esta importante propiedad, en la mayoría de aplicaciones la codificación ordinal no es especialmente eficiente. La causa está en que no se conserva la propiedad de localidad (ver apdo. 3.2.1), esto es, pequeñas variaciones en el código introducen cambios drásticos en los objetos codificados, y en consecuencia no existen verdaderos bloques constructivos, siendo la búsqueda es poco más que aleatoria. Eso se resume diciendo que el cruce, aun siendo un operador cerrado, carece de significado en el contexto del problema y por tanto no realiza un intercambio útil de información. Éste hecho alecciona al usuario para buscar un equilibrio entre la sencillez de la codificación —y los correspondientes operadores— y su significatividad para el problema que se pretende resolver.

5.2.3 Codificación y operadores de adyacencia

El mecanismo de codificación se sintetiza en lo siguiente: si un recorrido contiene la ruta $i \to j$ entonces se pondrá el número 'j' en la i-ésima posición del individuo que lo representa.

De ese modo, para el presente ejemplo, el recorrido

$$1 \to 2 \to 3 \to 4 \to 5 \to 6 \to 7 \to 8 \to 9 \hookleftarrow$$

se codificará así

$$\langle 2, 3, 4, 5, 6, 7, 8, 9, 1 \rangle$$

y para el otro recorrido, ie.,

$$4 \to 5 \to 2 \to 1 \to 8 \to 7 \to 6 \to 9 \to 3 \hookleftarrow$$

se tendrá la codificación

$$\langle 8_1, 1_2, 4_3, 5_4, 2_5, 9_6, 6_7, 7_8, 3_9 \rangle$$

Aunque no es necesario, se ha indicado la posición 'i' en los subíndices para facilitar la codificación.

Nótese que todos los individuos así codificados corresponden a tramos completos y cerrados, pero no necesariamente conexos. Por ejemplo, el individuo

$$\langle 2_1, 4_2, 8_3, 1_4, 9_5, 3_6, 5_7, 7_8, 6_9 \rangle$$

representa al recorrido

$$1 \to 2 \to 4 \hookleftarrow \qquad 3 \to 8 \to 9 \to 5 \to 7 \to 6 \hookleftarrow$$

que no es factible por tener dos componentes conexas (comúnmente llamadas *subrecorridos*).

Esta codificación es interesante porque, al contrario que la anterior, permite identificar de modo natural a los bloques constructivos. Por ejemplo, el esquema $(* * * 3 * 7 * * *)$ representa a todos los recorridos que contengan las rutas $4 \to 3$ y $6 \to 7$. Debido a esto es de esperar un buen rendimiento de los operadores genéticos que estén mínimamente bien construidos.

Se han desarrollado varios operadores de cruce que conservan la factibilidad para esta codificación. Los más usados son:

Cruce por alternancia de rutas: Se construye la descendencia seleccionando aleatoriamente rutas de uno y otro progenitor. Si la ruta recién introducida origina un subrecorrido se sustituye por alguna de las restantes que no lo origine.

Por ejemplo, el cruce de los individuos

$$\langle 2, 3, 8, 7, 9, 1, 4, 5, 6 \rangle \qquad \text{y} \qquad \langle 7, 5, 1, 6, 9, 2, 8, 4, 3 \rangle$$

proporciona estos descendientes

$$\langle 2,5,8,6,9,2,4,4,3\rangle \quad \text{y} \quad \langle 7,3,1,7,9,1,8,5,3\rangle$$

los cuales contienen errores en las posiciones 6ª y 8ª y 4ª, 6ª y 9ª respectivamente. Se corrigen esos errores sustituyendo las correspondientes ciudades por otras no presentes, por ejemplo

$$\langle 2,5,8,6,9,1,4,7,3\rangle \quad \text{y} \quad \langle 7,3,1,6,9,2,8,5,4\rangle$$

En la práctica este cruce tiende a estropear buenos subrecorridos, lo que reduce grandemente su aplicabilidad.

Cruce por alternancia de tramos: La idea es la misma de antes, pero alternando tramos de longitud aleatoria en vez de rutas sueltas. De esa manera se corrige la tendencia destructiva del cruce anterior, aunque a decir verdad sigue sin proporcionan buenas prestaciones.

Cruce heurístico: Se le dice heurístico porque toma en consideración los costes de cada ruta. Esto va en contra de la filosofía de los PEs, que recomienda utilizar en la medida de lo posible operadores ciegos; si se usa es porque aún no se han encontrado buenos operadores ciegos para esta representación que sean lo suficientemente generales.

Para construir un descendiente se toma al azar una ciudad como punto de partida y se comparan las rutas que salen de esa ciudad para uno y otro progenitor, eligiendo la de menor coste; la ciudad a que lleva la ruta elegida sirve como nuevo punto de arranque. El proceso se repite hasta retornar al punto de partida recorriendo todas las ciudades. Si en alguna etapa la nueva ruta origina un subrecorrido entonces se sustituye por otra de las restantes que no lo origine. Como todos los operadores heurísticos este es un operador específico para el PVC, por lo que ha sido necesario desarrollar muchas variaciones y particularizaciones dependiendo del problema en que se aplique; algunas de ellas se describen someramente en [MICH89,p215].

Como todos los operadores heurísticos —bien construidos, por supuesto— es mucho más eficaz que los operadores ciegos, debido a la información específica que se ha introducido. De hecho para PVCs con muchas ciudades, proporciona resultados comparables a los de las técnicas puramente heurísticas de resolución de dicho problema.

5.2.4 Mejoras a los Programas de Evolución combinatorios. Otras codificaciones

En [MICH94,p222 y ss.] se dan algunas ideas alternativas para resolver el PVC y en general otros problemas combinatorios a través de PEs. Seguidamente se da una descripción general de las más importantes.

Representaciones matriciales: La representación mediante vectores de números enteros parece ser todavía insuficiente para mostrar todas las propiedades importantes de un recorrido; por eso en algunas ocasiones es necesario servirse de tablas de conectividades y de posiciones relativas. Debido a esto, merece la pena plantearse la existencia de representaciones más ricas para el PVC y por extensión para los problemas combinatorios. Últimamente se están usando con notable éxito representaciones matriciales binarias de precedencias o de sucesores.

La *matriz de precedencias* es una matriz binaria $\mathbf{V} := [v_{ij}] \in \mathcal{R}^{m \times m}$ en la que el elemento v_{ij} vale 1 si y sólo si la ciudad i precede a la j y 0 en caso contrario. Tal matriz tiene ciertas propiedades que permiten construir operadores con sigunificado, tal y como se describe en [MICH94,p225].

La *matriz de sucesores* es una matriz binaria $\mathbf{W} := [w_{ij}] \in \mathcal{R}^{m \times m}$ en la que el elemento w_{ij} vale 1 si y sólo si el recorrido contiene la ruta $i \to j$ y 0 en caso contrario. De este modo, si el recorrido es completo y cerrado cada fila y cada columna de $\mathbf{W}$ contiene un uno y $m-1$ ceros, aunque no toda matriz con esas propiedades representa a un recorrido conexo. Eso es un inconveniente porque obliga a diseñar mecanismos de localización y reparación de subrecorridos, pero también es una ventaja porque existe la posibilidad de llegar a una buena solución pasando por un recorrido no conexo.

PEs combinatorios con búsqueda por perturbación local: Como ya se comentó en 5.1.3.2 es posible introducir en los PEs combinatorios el mecanismo de búsqueda por perturbación local, para ello se debe disponer de un buen algoritmo de construcción de 'soluciones perturbadas'. En [MICH94,p236] se referencian algunos intentos viables para construirlos. Muchos de los procedimientos de búsqueda por perturbación local son fuertemente heurísticos, de manera que lo que verdaderamente se está haciendo es una hibridación (ver 3.2.9).

Codificación mediante claves aleatorias: Se trata de resolver un problema basado en el orden utilizando un PE numérico. Para ello se codifica cada individuo como un vector de números reales, su ordenación (por ejemplo, de menor a mayor) señalará la solución asociada. La ventaja estriba en que se pueden utilizar todos los operadores específicos de los PEs numéricos. De hecho, esta codificación se puede utilizar con otros paradigmas de la computación evolutiva (ver [MICH94,p222]).

5.2.4.1 El problema de la planificación de horarios

El problema de la planificación de horarios es fundamental en Investigación Operativa, tanto a nivel práctico como teórico (ver [MICH94,p256] y [HEIT96,Q2]). Resulta imposible describirlo con detalle en un espacio reducido, pese a lo cual se introducirá someramente por ser uno de los ejemplos más destacados de PEs con representación matricial.

A grandes rasgos, el problema de planificación de horarios se expresa así:

En un taller se dispone de m operarios $\{T_1, \ldots, T_m\}$ que deben atender k puestos $\{C_1, \ldots, C_k\}$ durante n turnos $\{H_1, \ldots, H_n\}$ a lo largo de la semana —vg., H_1 representa el turno “lunes por la mañana”.
Se trata de obtener el horario más adecuado —con arreglo a unos objetivos especificados— que satisfaga ciertas restricciones del estilo de:

- En todo momento debe haber un solo operario en cada puesto.
- Un operario no puede estar en más de un puesto a la vez, ie., sólo puede estar en un único puesto en cierto turno.
- El número de operarios desocupados debe ser lo menor posible.

y otras restricciones más flexibles de naturaleza diversa, tales como:

- Cada operario tiene estipulado un número de horas a la semana. No es conveniente (a veces es imposible) desviarse mucho de ese número.

Así planteado, el problema se puede resolver mediante un PE que contenga los siguientes procedimientos:

Función de aptitud: Es habitual construirla como la suma de las *productividades* parciales obtenidas por los m operarios más un término global que mide el efecto sinergético del conjunto.

Representación: La más natural consiste en una matriz de enteros $\mathbf{R} := [r_{ij}] \in \mathcal{R}^{m\times n}$ en la que el elemento $r_{ij} \in \{C_1, \ldots, C_k\}$ señala el puesto que le ha correspondido al operario T_i en el turno H_j, es decir

	H_1	H_2	$\ldots$	H_n
T_1	r_{11}	r_{12}	$\ldots$	r_{1n}
T_2	r_{21}	r_{22}	$\ldots$	r_{2n}
$\vdots$			$\ddots$	$\vdots$
T_m	r_{m1}	r_{m2}	$\ldots$	r_{mn}

Operadores genéticos: Hay una gran cantidad de ellos; los más comunes son:

- **Mutación de orden *p*:** Para un solo operario (fila) se toman dos sucesiones contiguas de p turnos y se intercambian.
- **Mutación completa de orden *p*:** Es la misma de antes, sólo que se hace para todos los operarios en vez de para uno solo.

- **Cruce heurístico de orden *q*:** En cada progenitor se calcula para cada operario (fila) la aptitud local asociada al horario asignado y se intercambian los q mejores con el otro progenitor. El parámetro q suele ir variando a lo largo de la búsqueda.

Algoritmos de reparación: Con ellos se tratan de evitar soluciones inconsistentes. Su diseño suele consumir la mayor parte del esfuerzo de implantación, ya que suelen ser fuertemente heurísticos.

El problema de planificación de horarios no es más que un caso particular de otro problema mucho más general que se denomina *problema de la planificación de tareas*. Dicho problema se describe de modo genérico en [HEIT96,Q2] y ha sido resuelto eficazmente por varios autores sirviéndose de técnicas evolutivas (ver, vg., [DAVI91,p332] y [PROC89,p160]); posiblemente el planteamiento evolutivo más claro y sencillo para este problema es el propuesto por S. Uckun *et al.* en [UCKU93].

5.2.4.2 Otros problemas combinatorios resueltos mediante PEs

El capítulo 9 de [MICH94] está específicamente dedicado a la resolución detallada de varios problemas combinatorios típicos a través de PEs. concretamente se resuelven los problemas de *trazado de grafos* y de *partición de conjuntos*, y se plantean pormenorizadamente un problema de *planificación de tareas* y otro de *cálculo de trayectorias* de un vehículo en un entorno desconocido. Debido a la amplitud del tratamiento y a la cantidad de sugerencias que se proponen para futuros desarrollos constituye una referencia obligada para quien quien desee resolver problemas combinatorios no triviales mediante PEs.

5.3 Conclusiones

En este capítulo se han descrito los PEs como métodos útiles de incorporar directamente conocimiento específico a los AGs. Se ha comprobado cómo la utilización de estructuras de datos naturales y operadores genéticos sensibles al contexto mejora grandemente la eficiencia de los AGs sin perder demasiada generalidad. Además, si se hace adecuadamente dicha incorporación, el conocimiento específico que se incorpora es muy genérico, por lo que no se incrementa mucho el esfuerzo de implantación.

En la práctica no es fácil "hacer adecuadamente dicha incorporación", requiriéndose un considerable estudio previo de la naturaleza del problema. Además, la cuestión de cuánto conocimiento específico añadir en cada caso es especialmente delicada porque depende de circunstancias externas al problema, como los equipos disponibles, la rentabilidad del desarrollo, existencia de métodos alternativos de resolución,etc. En general, para construir un PE se deben estudiar detenidamente estos cinco mecanismos básicos:

- Elección de una representación del dominio del problema que sea natural y sencilla de manejar y que a la vez proporcione bloques constructivos con significado.
- Diseño (meticuloso) de los operadores genéticos: deben tener significado para la representación actual y tratar adecuadamente las restricciones (sensibilidad al contexto).
- Elección una función de evaluación que distribuya ordenadamente las soluciones en función de su grado de idoneidad (ie., aptitud).
- Determinación de la población inicial de candidatos a solución. Es conveniente que alguno de ellos sea factible.
- Parametrización del PE. Este proceso es aún más crítico que en los AGs debido a los requerimientos de eficacia más estrictos y a la mayor cantidad de parámetros a ajustar.

Para implantar los cinco mecanismos resulta muy conveniente aprovechar la experiencia de otros PEs. De hecho, puesto que no es nueva la necesidad de incorporar conocimiento específico en los AGs, se pueden extraer sustanciosas conclusiones de AGs aplicados a problemas concretos del ámbito empresarial, aunque no sigan el paradigma de los PEs. Concretamente, toda la segunda parte del libro [DAVI91] y muchos de los artículos de las partes segunda y tercera de [PROC89] están dedicadas a la resolución de complicados problemas científicos y técnicos a través de Algoritmos Genéticos. En aquel tiempo aún no había sido desarrollado por completo el paradigma de los Programas Evolutivos, por lo que se echa en falta un tratamiento unificado de los problemas. A pesar de ello, se proporciona una apreciable cantidad de excelentes ideas sobre los operadores y las representaciones aplicables en campos muy diversos.

Capítulo 6

Las Estrategias Evolutivas

Las estrategias evolutivas (en lo sucesivo ES, de *Evolutionstrategien*) surgieron como métodos estocásticos de escalada con paso adaptativo específicamente diseñados para resolver problemas de optimización paramétrica. Con el paso del tiempo fueron incorporando procedimientos propios de la computación evolutiva hasta convertirse en un paradigma más de dicha metodología. En el presente, las ES son algoritmos evolutivos enfocados preferentemente hacia la optimización paramétrica que utilizan una representación a través de vectores reales, selección determinista y operadores específicos de mutación y cruce, el cual ocupa ahora un lugar secundario con respecto a la mutación. Las ES son el paradigma más fuerte (ie., más específico) de la CE; sin embargo, mantienen la filosofía de las 'cajas negras' y operadores ciegos.

6.1 Estrategias Evolutivas simples

Las ES simples —también llamadas *ES de dos miembros*— son el modelo más sencillo de su paradigma. Aunque no son métodos evolutivos propiamente dichos contienen en su forma más pura gran parte de las ideas que incorporan las ES múltiples. Pese a su sencillez, las ES simples tienen utilidad práctica y aún se siguen utilizando versiones mejoradas para resolver 'problemas reales' de ingeniería, como seguidamente se verá.

Las ES simples son básicamente procedimientos estocásticos de optimización paramétrica con paso adaptativo. En la terminología del ramo se denotan como $(1+1)$-ES y se describen así:

> En las $(1 + 1)$-ES evoluciona un sólo individuo haciendo uso únicamente de un operador genético, la mutación. Los individuos son bicromosomáticos y se representan a través de un par de vectores reales
>
> $$\mathbf{v} := \langle \mathbf{x}; \mathbf{s} \rangle$$

> El primer vector $\mathbf{x}$ representa a un punto del espacio de búsqueda, y el segundo, $\mathbf{s}$, es un vector de desviaciones típicas que se usa al realizar la mutación. Al haber un sólo progenitor no hay selección, y el criterio de reemplazo consiste en que el descendiente reemplaza a su progenitor si y sólo si es más apto que éste; en caso contrario, el descendiente es eliminado y la "población" permanece sin cambios.

Concretamente, la evolución del cromosoma $\mathbf{x}$ se puede sintetizar así:

$$\mathbf{x}[t+1] = \begin{cases} \mathbf{x}[t] + \mathbf{N}(0, \mathbf{s}[k]) & \text{sii } \mathbf{x}[t+1] \succ \mathbf{x}[t] \quad \wedge \quad \mathbf{x}[t+1] \in \mathcal{X} \\ \mathbf{x}[t] & \text{en caso contrario} \end{cases}$$

donde $\mathbf{N}(0, \mathbf{s})$ es un vector de números aleatorios gaussianos independientes con medias 0 y desviaciones típicas $\mathbf{s}$; a veces se también se usa una distribución de probabilidades lognormal. Salvo indicación en contra habitualmente se acepta que

$$\mathbf{s}[t] := \sigma[t] \cdot (1, \ldots, 1)$$

Puede demostrarse (ver [MICH94,p169] y [VOIG91]) que para óptimos finitos y bajo ciertas condiciones de regularidad el esquema anterior converge al óptimo cuando $t \to \infty$ con probabilidad 1. No existen resultados concluyentes sobre la velocidad de convergencia, tan solo una regla semiempírica que afirma lo siguiente:

> *Hipótesis de* 1/5 *de éxitos:* La velocidad óptima de convergencia se alcanza cuando la relación de mutaciones con éxito frente al total de mutaciones, $\varphi[t]$, vale 1/5.

A consecuencia de esa hipótesis resulta innecesario someter a evolución al segundo cromosoma, dado que ya se conoce la estrategia óptima de evolución para él: $\mathbf{s}^o[t]$ es tal que proporciona una tasa de éxito en las mutaciones lo más próxima posible a 1/5. Eso se concreta en la siguiente ley de adaptación para el segundo cromosoma:

$$\sigma[t+1] = \begin{cases} c_d\, \sigma[t] & \text{sii} \quad \varphi_k[t] > 1/5; \\ \sigma[t] & \text{sii} \quad \varphi_k[t] \simeq 1/5; \\ c_i\, \sigma[t] & \text{sii} \quad \varphi_k[t] < 1/5. \end{cases}$$

donde $\varphi_k[t]$ es el coeficiente de éxito del operador mutación durante las k últimas generaciones (k es un parámetro del método) y $c_i > 1$ y $c_d < 1$ son coeficientes con los que se regula la tasa de adaptación. Una parametrización típica es:

$$k \simeq 5 \qquad c_d \simeq 0.82 \qquad c_i = 1/c_d$$

De ese modo, la búsqueda avanzará con pasos largos cuando tenga éxito y con pasos cortos en caso contrario.

Nótese que la mutación propiamente dicha sólo afecta al primer cromosoma, pues el segundo tan sólo se adapta en función del éxito que haya tenido la mutación del primero.

En definitiva, las $(1+1)$-ES responden al siguiente esquema genérico:

```
BeginAlgorithm{Evolutionstrategie}
 CntSuccess = 0;   SuccessRate = 0;
 Xcur = InitX();   Fcur = Fobj(Xcur);
 Sigma = InitSigma();
 for( t = 0 ; t < MaxNumGen ; t++) {
  Scur = Sigma * Ones(m);
  Xnew = Xcur + RandN(Zeros(m), Scur);
  Fnew = Fobj(Xnew);
  if( (Fnew > Fcur) && TestBounds() ) {
   Xcur = Xnew;   Fcur = Fnew;
   CntSuccess++;
  }
  SuccessRate = CntSuccess / t;
  if (t % Tadap) {                     Actualiza las dispersiones ca-
                                       da Tadap pasos
   if SuccessRate < 0.2
    Sigma = Sigma * Adapt;
   elseif Successrate > 0.2
    Sigma = Sigma / Adapt;
  }
 }
EndAlgorithm
```

Se han desarrollado muchas mejoras para ese esquema básico; la mayoría de ellas son sofisticaciones del operador de mutación o del proceso de adaptación de las desviaciones.

6.2 Estrategias Evolutivas múltiples

Al igual que el resto de métodos de búsqueda simple, las $(1+1)$-ES tienden —de forma más o menos acusada— a converger hacia subóptimos; para corregir esa tendencia es conveniente realizar una búsqueda múltiple y una selección de los mejores descendientes. Ahora bien, la mutación por sí sola no tiene la capacidad de recombinar información de buenos individuos; por eso también es necesario introducir operadores de cruce. Así es como surgieron las *ES múltiples* o *multimembres.* Técnicamente se denotan como $(\lambda \overset{+}{,} \mu)$-ES, indicando λ el tamaño de la población y μ el tamaño de la descendencia; el símbolo "$\overset{+}{,}$" se usa para señalar la existencia de dos posibilidades de reemplazo: reemplazo determinista por inclusión (o de tipo 'más') o reemplazo determinista por inserción (o de tipo 'coma').

Las $(\lambda \overset{+}{,} \mu)$-ES incorporan los siguientes métodos y criterios:

Criterio de representación: Es el ya comentado, ie., mediante parejas de vectores reales. No es infrecuente usar una representación mediante tríos, en la que el último vector controla la correlación entre las mutaciones de cada componente.

Criterio de tratamiento de los individuos no factibles: Como el anterior, es decir, se ignoran los individuos no factibles (filtrado).

Operadores genéticos: Son por defecto el cruce y la mutación, aunque se pueden introducir otros.
• En lo que respecta al cruce, se usa habitualmente el cruce uniforme (para ambos cromosomas) o el *cruce intermedio*, que consiste en generar un único descendiente promediando los dos progenitores $\langle(x_1,\ldots,x_m);(s_1,\ldots,s_m)\rangle$ y $\langle(x'_1,\ldots,x'_m);(s'_1,\ldots,s'_m)\rangle$ así:

$$\left\langle\left(\frac{1}{2}(x_1+x'_1),\ldots,\frac{1}{2}(x_m+x'_m)\right);\left(\sqrt{(s_1)^2+(s'_1)^2},\ldots,\sqrt{(s_m)^2+(s'_m)^2}\right)\right\rangle$$

En ocasiones se realiza el cruce en *modo global*, esto es, tomando un nuevo par de progenitores para cada componente de la descendencia.
• La mutación es la misma de antes, ie.,

$$\mathbf{x}[t+1] = \mathbf{x}[t] + \mathbf{N}(0, \mathbf{s}[t])$$

salvo que, como ahora no se verifica la hipótesis del 1/5 de éxitos, el segundo cromosoma se somete también a evolución (mutación), de la siguiente manera:

$$\mathbf{s}[t+1] = e^{N(0,\tau)} \cdot \mathbf{s}[t]$$

donde τ es un parámetro del método.

Criterio de selección: Cualquier miembro de la población puede ser elegido como progenitor con igual probabilidad (muestreo aleatorio simple).

Criterios de reemplazo: En las ES el reemplazo siempre es determinista: se eligen los μ mejores miembros. Existen dos posibilidades de hacer esa elección, por inserción o por inclusión (ver 2.1.3), lo que da lugar a dos variedades de ES múltiples, las (λ,μ)-ES y las $(\lambda+\mu)$-ES.
Reemplazo por inclusión: Se juntan los μ descendientes con los λ criadores en una sola población, y de ella se toman los λ mejores miembros para la nueva población. Las ES que usan este tipo de reemplazo se denotan $(\lambda+\mu)$-ES.
Reemplazo por inserción: Según el tamaño relativo de la descendencia respecto de la población se distinguen dos casos:
• $\mu \leq \lambda$: Se eliminan los μ peores miembros de la población de criadores. Su lugar será ocupado por los μ descendientes.

• $\mu > \lambda$: Se constituye la nueva población tomando los λ mejores miembros de la población de descendientes.
Las ES que usan este tipo de reemplazo se denotan (λ, μ)-ES. Nótese que en las (λ, μ)-ES la población varía mucho de una generación a otra. Debido a ello deben usarse con especial cuidado y cuando se obtenga algún beneficio de esa variabilidad, vg., al optimizar objetivos no variables en el tiempo o 'ruidosos'.

Se pueden incorporar muchas mejoras en $(\lambda \overset{+}{,} \mu)$-ES y crear nuevas variedades modificando los anteriores métodos y criterios. Algunas de esas modificaciones se describen en [VOIG91]. En [MICH94,p222] se da noticia de una nueva variedad de ES que permite resolver directamente problemas combinatorios. Dicha variedad utiliza una representación mediante vectores enteros y nuevos operadores de mutación adaptados a esa representación.

6.3 Comparación de las ES con los AGs

Aunque en su origen las ES y los AGs partieron de enfoques distintos, los sucesivos desarrollos de ambos se han ido acercando hasta el punto de que, actualmente, una ES se puede considerar como un tipo especial de AG o viceversa, distinguiéndose sólo en el uso de unos u otros procedimientos por defecto. De todos modos, no está de más recordar las diferencias existentes entre ambos y las consecuencias prácticas que conllevan:

⋆ Las ES son intrínsecamente más fuertes que los AGs, ie., requieren para funcionar más conocimiento específico que los AGs, los cuales son, por construcción, algoritmos de búsqueda de propósito general.

⋆ El campo natural de aplicación de las ES son los problemas de optimización paramétrica, el de los AGs la optimización de atributos. Ello no quiere decir que no se puedan usar las ES para resolver problemas combinatorios; siempre se pueden resolver indirectamente, utilizando una codificación como la de las claves aleatorias (ver apdo. 5.2.4), o directamente mediante ES específicas.

⋆ En los AGs la clave de la búsqueda está en los operadores genéticos de recombinación (cruce), en las ES en los de alteración (mutación).

⋆ Por defecto, el criterio de reemplazo en los AGs es estocástico, en las ES determinista. En uno y otro caso se trata de mantener un nivel óptimo de aleatoreidad frente a determinismo.

⋆ En las ES el tamaño de la descendencia está fijado de antemano, en los AGs sólo está fijado su promedio (a través de la tasa de cruce). Además, en las ES todos los individuos son sometidos a mutación, la cual es más drástica para unos individuos y menos para otros —según cómo sea el vector de dispersiones; en los AGs sólo unos pocos individuos son sometidos a mutación, la cual es igual de drástica para todos ellos.

⋆ El tratamiento de los puntos no factibles se lleva a cabo en las ES implícitamente mediante filtrado; en los AGs el tratamiento de la no factibilidad requiere un tratamiento específico. Además, en los AGs la técnica de filtrado de las soluciones no factibles no es usual ni provechosa. De hecho, esa técnica se usa muy poco; en las ES da buenos resultados debido a la capacidad de autoajuste de los parámetros evolutivos.

⋆ Los AGs son mecanismos de búsqueda preferentemente global, esto es, tienen dificultades para el ajuste fino; con las ES sucede justo al contrario.

Todas estas diferencias motivan que, aun compartiendo mecanismos básicos, unas y otros tengan campos específicos de aplicación.

6.4 Minimización mediante ES de los costes de seguimiento de procesos dinámicos

Para ilustrar la potencia y flexibilidad de las Estrategias Evolutivas se va a resolver completamente un problema "real" de la Teoría de Sistemas. El uso de técnicas evolutivas permite utilizar una formulación genérica, esto es, sin información específica; de este modo resulta inmediato particularizar las soluciones obtenidas a una gran variedad de sistemas que se dan en el ámbito empresarial.

Dado un proceso dinámico realimentado como el mostrado en la figura 6.1 se trata de diseñar un control que permita minimizar los costes ocasionados por la planta al tratar de seguir la señal de referencia.

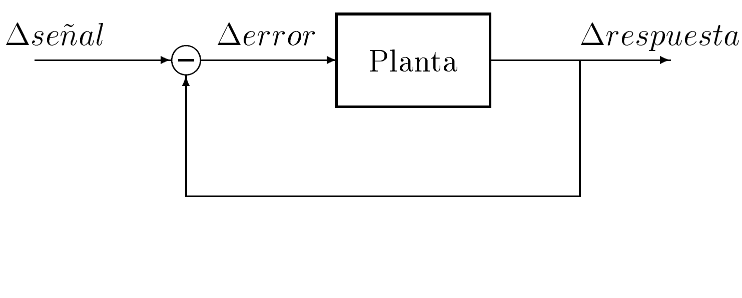

Figura 6.1: Modelo de seguimiento

El modelo de referencia es genérico (modelo de cajas negras), y en el ámbito empresarial cabe hacer la siguiente particularización entre otras muchas: la planta es una factoría o almacén, la señal de referencia, $u[k]$, es una demanda de cierto producto y la respuesta, $y[k]$, es la oferta que es capaz de generar dicha planta. En condiciones de equilibrio la oferta se adecúa a la demanda y el error, $e[k] := u[k] - y[k]$ es nulo.

Al producirse una variación en la demanda, $\Delta u[k]$, la planta no puede seguirla instantáneamente, lo que ocasiona un error en el seguimiento de la demanda $\Delta e[k] \neq 0$, tal y como se aprecia en la figura 6.2.

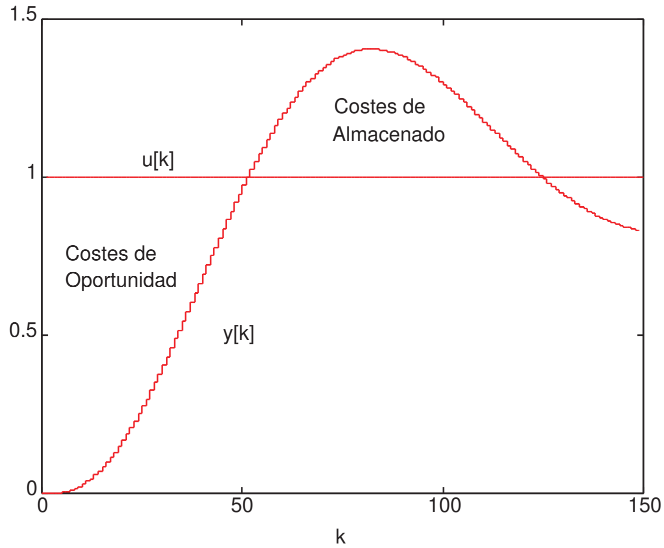

Figura 6.2: Respuesta dinámica

Para la planta esos errores son perjudiciales dado que cuando $\Delta e < 0$ la oferta supera a la demanda y se incurre en costes de almacenamiento, recíprocamente cuando $\Delta e > 0$ la demanda supera a la oferta y se incurre en costes de oportunidad.

Se trata, pues, de diseñar un modelo de gestión (figura 6.3) que corrija la respuesta de tal manera que haga mínimo el valor de los costes totales de seguimiento, medidos a través del siguiente índice

$$C_T := \sum_{k=0}^{\infty} c_{al}[k] \cdot \mathcal{R}(e[k]) + c_{op}[k] \cdot \mathcal{R}(-e[k])$$

donde $c_{al}[k]$ y $c_{op}[k]$ dan los valores de los costes de almacenamiento y oportunidad en cada instante k y $\mathcal{R}(z)$ es la función rampa, ie.,

$$\mathcal{R}(z) := \begin{cases} z & \text{sii } \; z > 0 \\ 0 & \text{en caso contrario} \end{cases}$$

Para ajustarse lo más posible a la realidad conviene imponer las siguientes restricciones al modelo de gestión

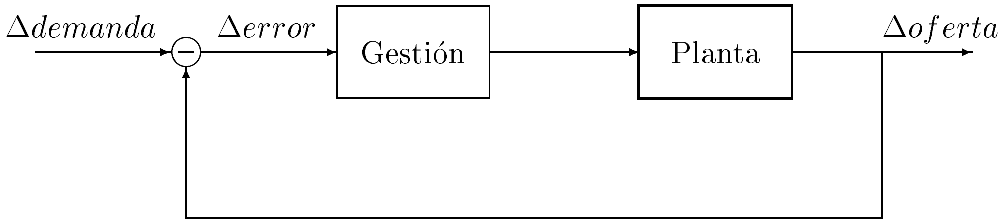

Figura 6.3: Modelo de seguimiento corregido

- La introducción de la gestión no debe alejar a la planta del seguimiento exacto de la señal en estado estacionario, es decir, se debe seguir verificando que
$$\lim_{k\to\infty} e[k] = 0$$
- La gestión no puede someter a la planta a seguir un comportamiento fuertemente variable. Los 'sistemas reales' y más aún los que surgen en el ámbito empresarial evolucionan "suavemente".

Admitiendo que el proceso de la planta puede ser identificado a través de un modelo (discreto) de ecuación en diferencias $\mathcal{P}(q)$ —donde q es el operador de retraso—, la teoría clásica de sistemas permite calcular un modelo de gestión $\mathcal{G}(q)$ qua haga mínima una función de costes cuadrática, ie., $C_T = \sum_k (e[k])^2$, pero no es posible hacer lo mismo para una función de costes como la indicada, y menos aún si se imponen restricciones de comportamiento (ver [PHIL89]).

Es aquí donde cabe hacer uso de las ES debido a que el problema es esencialmente numérico, pero se quiere conservar la mayor generalidad posible para que siga siendo válido el modelo de cajas negras.

Con el fin de ilustrar cómo se implanta una ES para resolver problemas de este tipo, se ha programado en el lenguaje de alto nivel de SCILAB —émulo de dominio público del programa MATLAB— una estrategia evolutiva simple con diversas modificaciones específicas que reducen la tendencia a la convergencia prematura y/o al extravío.

Sea el siguiente modelo de planta, identificado mediante una herramienta de análisis de series temporales,
$$\mathcal{P}(q) = 4.014 \times 10^{-5} \frac{q + 3.8534q^2 + 0.9276q^3}{1 - 2.8561q + 2.7168q^2 - 0.8607q^3}$$
y el índice de costes que se deriva de $c_{al}[k] = c_{op}[k] \equiv 1$, ie.,
$$C_T = \sum_{k=1}^{\infty} |e[k]|$$

Cuando no hay gestión $\mathcal{G}(q)$, al producirse una variación unitaria en la demanda la oferta evoluciona según la gráfica mostrada en la figura 6.4, lo que lleva al sistema a incurrir en unos costes totales que se han evaluado en $C_T = 54.1472$ unidades monetarias.

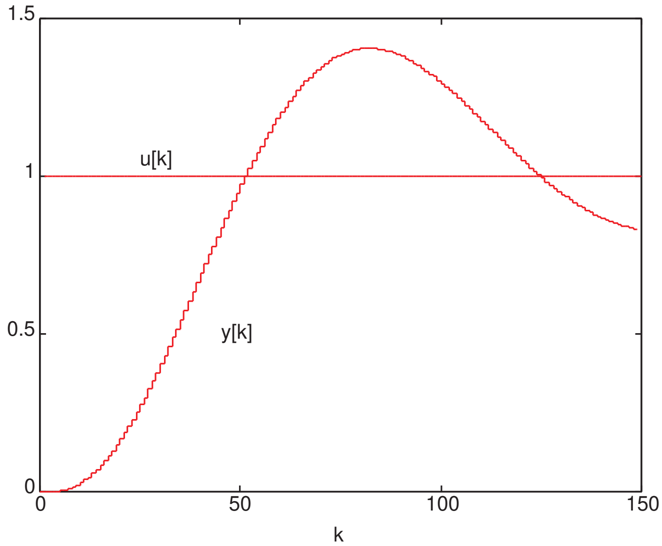

Figura 6.4: Respuesta sin corrección

Para reducir dichos costos de seguimiento se ha propuesto un modelo de gestión de segundo orden así

$$\mathcal{G}(q) := \frac{a_0 + a_1 q + a_2 q^2}{1 + b_1 q + b_2 q^2}$$

Se trata de hallar los valores de los coeficientes a_0, a_1, a_2, b_1 y b_2 que minimicen C_T sin violar las restricciones, que se establecen en $a_i, b_i \in [-200, +200]$.

Tras varios ensayos de tanteo se han elegido los siguientes valores iniciales

$$a_0 = 30 \quad a_1 = -60 \quad a_2 = 25 \quad b_1 = 1 \quad b_2 = -1$$

con ellos se ejecuta la rutina `ESTrack`, la cual ha proporcionado el siguiente resultado:

```
Resultados del paso 20:50
     a_0             34.3901
     a_1            -57.5219
     a_2             23.4626
     b_1             -1.6693
     b_2             -0.8177
    COSTES           11.3580
```

Al ejecutar el programa se observa que en las primeras etapas de la búsqueda los controles hallados originan costes altísimos (del orden de 10^{13}), eso es señal de que son inestables o bien de que no pueden mantener el requisito de seguimiento exacto. Al ir progresando la búsqueda, la ES proporciona controles estables y de buen comportamiento dinámico que además logran reducir los costes de seguimiento hasta poco más de la quinta parte.

Ahora la oferta responde a una variación unitaria de la demanda a la manera indicada en la figura 6.5.

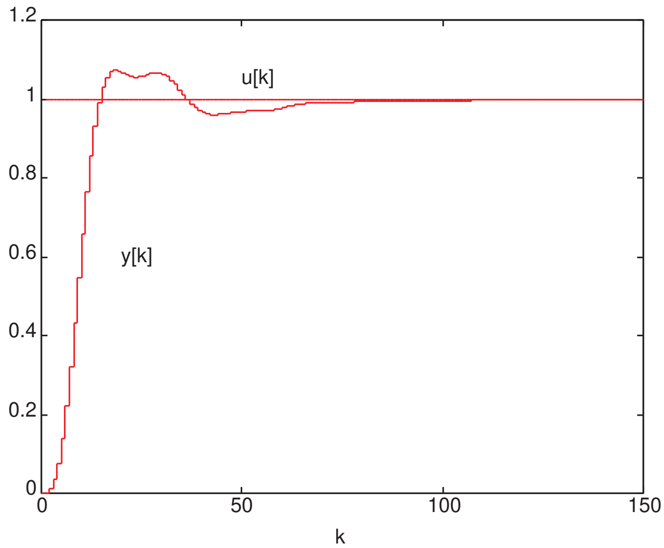

Figura 6.5: Respuesta corregida

La rutina `ESTrack` se pude generalizar de modo inmediato para que trabaje con funciones de costes más complejas y modelos de gestión de mayor orden. De hecho no es difícil modificarla para que funcione con modelos completamente opacos de planta y gestión. En el apéndice B se muestra su listado.

Capítulo 7

Sistemas de Aprendizaje Artificial basados en AEs

Hasta el momento, el objeto principal de la presente memoria ha consistido en estudiar las posibilidades que ofrecen los algoritmos evolutivos en la resolución de complejos problemas de búsqueda que surgen en el ámbito empresarial. Ciertamente, muchos de los problemas clásicos de la ingeniería admiten fácilmente un planteamiento como problemas de búsqueda, por lo que es del máximo interés disponer de técnicas lo más generales posibles para resolverlos. Sin embargo, en las últimas décadas ese planteamiento ha resultado ser excesivamente rígido para plasmar correctamente problemas que surgían en campos punteros de la ingeniería tales como la Inteligencia Artificial, o la Teoría de Decisiones. No es que esos nuevos problemas conlleven búsquedas difíciles, es que difícilmente se pueden plantear como problemas de búsqueda. Considérese, por ejemplo, el problema del pilotado de un vehículo en un entorno variable. Es posible plantear ese problema como uno de control óptimo, pero —independientemente de lo difícil que sea su resolución— tal planteamiento introducirá inevitablemente simplificaciones inaceptables o será demasiado específico como para resolver un amplio espectro de contingencias. Parece claro que se precisaría un planteamiento más abierto y flexible que permitiera al vehículo adquirir pautas de comportamiento basándose en experiencias previas. A este planteamiento se le llama ***aprendizaje artificial*** (en la bibliografía inglesa, *machine learning*), dado que a la larga se trata de reproducir en sistemas formales los procesos de aprendizaje de los seres vivos.

Los enfoques más utilizados para resolver los problemas así planteados son los basados en Sistemas Expertos y en Redes Neuronales. La resolución de problemas de aprendizaje artificial a través de Algoritmos Evolutivos no sólo es posible, sino que en determinados aspectos es netamente ventajosa sobre las anteriores. Su implantación no está exenta de inconvenientes —de hecho surgen algunos problemas técnicos de solución delicada—, pero a la larga tales problemas no son mayores que los planteados por los otros enfoques.

7.1 Sistemas clasificadores (Formulación de Michigan)

La implantación más común de sistemas de aprendizaje artificial basados en AEs se hace a través de los ***sistemas clasificadores***. Formalmente, un sistema clasificador es un sistema de aprendizaje inductivo de reglas de decisión sobre espacios basados en atributos con el que se trata de producir respuestas adecuadas a un conjunto de estímulos a través de un proceso de clasificación de conceptos que van siendo adquiridos empíricamente.

Tradicionalmente se ha venido utilizando una metáfora biológica para mostrar el funcionamiento elemental de un sistema clasificador, es el 'Sistema Clasificador Kermit'. Kermit es una rana que vive en un entorno perfectamente delimitado (una charca) en el que recibe ciertos estímulos y efectúa ciertas respuestas. Los estímulos son recibidos fundamentalmente a través de sus ojos, los cuales son tan simples que sólo le permiten percibir cinco atributos de los objetos del entorno, a saber: si el objeto es móvil o inmóvil, si vuela o está en el suelo, si es grande o pequeño, si está cerca o lejos y si tiene colores vivos o apagados. El conjunto de respuestas que Kermit puede efectuar es también bastante limitado: puede huir, puede camuflarse o puede intentar comerse el objeto en cuestión. De lo que se trata es de proporcionar a Kermit un conjunto mínimo de reglas de decisión que le permitan sobrevivir con éxito en su entorno; esas reglas serán del estilo de "Si el objeto es grande y se mueve, huir" o bien "Si el objeto es pequeño, se mueve, está en el aire y cerca, comérselo". Es evidente que el conjunto de reglas ha de ser lo más reducido posible porque una rápida respuesta puede salvarle la vida, además ese conjunto de reglas ha de ser lo suficiente general como para considerar todas las situaciones importantes que plantee la vida en una charca, y a la vez no debe ser ambiguo. Nótese que Kermit no tiene por qué saber lo que es un mosquito o una garza, simplemente tiene que tener muy claros los conceptos de peligrosidad y comestibilidad y saber clasificar los estímulos que le llegan en función de esos conceptos. Las ranas reales han adquirido ese conjunto de reglas con el que hacer la clasificación a través de un largo proceso de evolución natural; parece pues bastante razonable utilizar la evolución artificial para emular el proceso de aprendizaje.

Como acaba de verse, los sistemas clasificadores (abreviadamente, SCs) son estructuras computacionales basadas en reglas de decisión del estilo de

```
SI    < condicion >    ENTONCES   < accion >
```

que poseen mecanismos generales para procesar las reglas en paralelo, para generar nuevas reglas de modo evolutivo y para verificar su eficacia en el entorno. Eso se logra manteniendo una población de reglas codificadas como cadenas de bits —precisamente, es a dichas cadenas a lo que se llama *clasificadores*— que evolucionan en función de una serie alterna de estímulos y refuerzos proporcionados por el entorno. La finalidad última es que el sistema aprenda de modo inductivo un conjunto de conceptos que le permita responder apropiadamente a los estímulos del entorno.

El esquema genérico de un SC es el que se muestra en la figura.

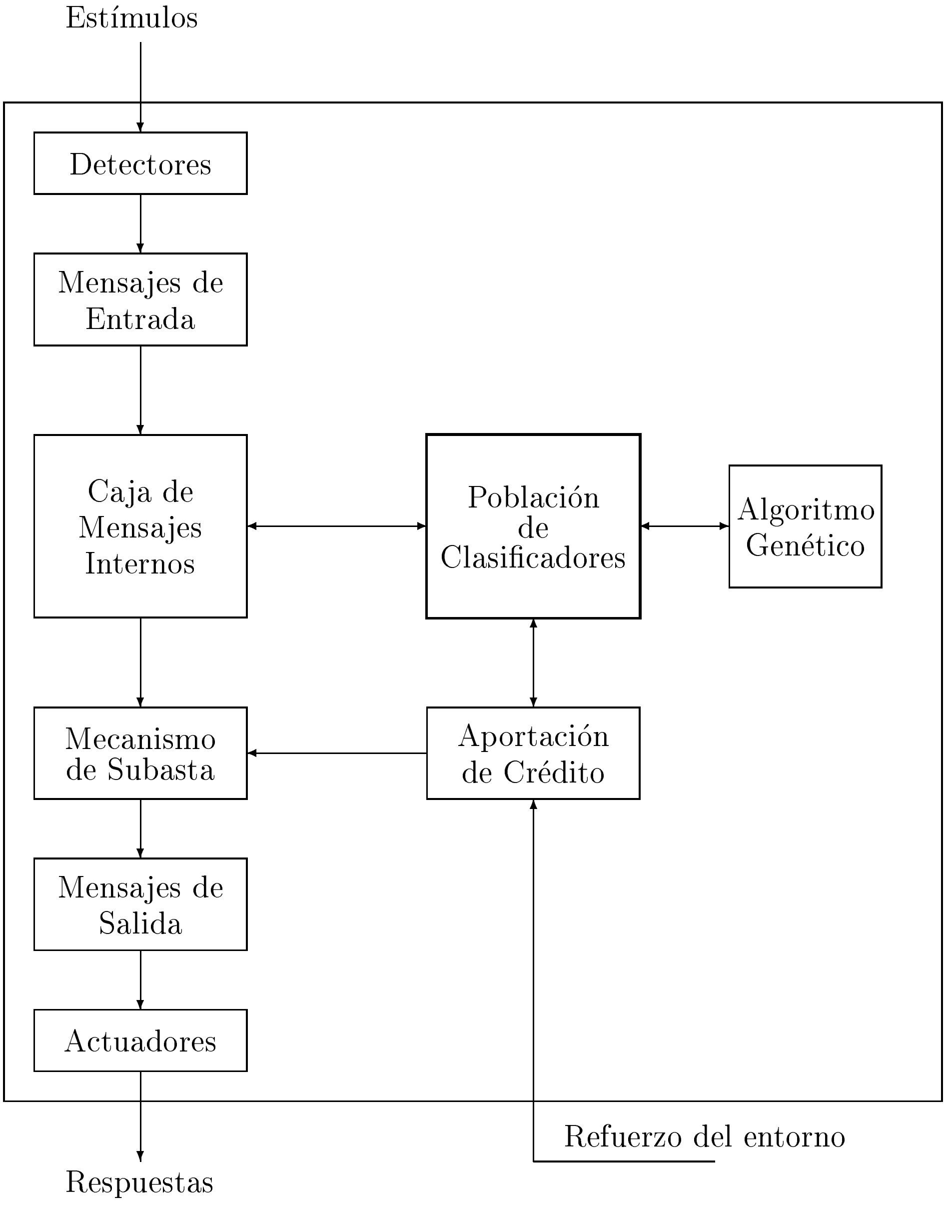

Figura 7.1: Estructura genérica de un SC

Su funcionamiento a grandes rasgos es como sigue.
El entorno provoca la activación de los detectores, que es procesada en forma de estímulos o *mensajes de entrada*, los cuales van pasando a una *caja de mensajes*

internos. El SC mantiene también una población de reglas, algunas de las cuales serán activadas a su debido tiempo por los mensajes internos. No todas las respuestas producidas darán lugar a *mensajes de salida* sino que se establece una *subasta* en la que las diversas reglas activadas pujan por mandar sus mensajes a los *actuadores*, de tal suerte que sólo las reglas más *fuertes* conseguirán hacerlo; así se evita producir órdenes contradictorias y se agiliza el procesamiento de respuestas. El *mecanismo de aportación de crédito* se encarga de actualizar las fuerzas de cada regla repartiendo entre ellas el *refuerzo* proporcionado por el entorno de modo que se favorezca a las mejores. Por último, cada cierto número de ciclos se pone en marcha un procedimiento evolutivo de actualización de la población de reglas, cuya tarea no es tanto encontrar la mejor regla como la de proporcionar una población completa de ellas que ejecute lo mejor posible la labor de reconocimiento y clasificación de conceptos.

Muchos sistemas de aprendizaje artificial representan el conocimiento a través de reglas de decisión; sin embargo, los sistemas clasificadores tienen ciertas peculiaridades que los distingue del resto. Siguiendo a Goldberg [GOLD89,p257],

> Al igual que ocurre con otros sistemas basados en reglas, las reglas de un sistema clasificador son de la forma "SI $<$ *condición* $>$ ENTONCES $<$ *acción* $>$"; sin embargo, en los sistemas clasificadores las condiciones y las acciones están obligadas a ser cadenas de longitud fija a las que se proporciona la capacidad de reconocimiento de patrones incluyendo un signo de indiferencia. Además, los sistemas clasificadores se apartan de la corriente principal de los sistemas expertos al permitir la activación paralela de reglas. Eso relaja el cuello de botella que surge en los sistemas de procesado secuencial de reglas, permitiendo con ello múltiples razonamientos y acciones. Cuando los sistemas clasificadores deben arbitrar entre dos alternativas mutuamente excluyentes en competencia, se apartan también de la corriente principal de los sistemas expertos utilizando estrategias competitivas de arbitrio en vez de procedimientos basados en el orden o similares.

Un ejemplo práctico servirá para ilustrar todo lo visto hasta ahora.

7.1.1 Ejemplo de aprendizaje de un concepto a través de un SC

Considérese el problema de controlar una cuba de fermentación. En la cuba se pueden medir u observar seis atributos: la presión (P), la temperatura (T), el caudal (Q), la densidad (D), la turbidez (U?) y la ebullición (V?). Como la fermentación es un proceso fuertemente no estacionario, sólo se pueden cuantificar tres rangos de valores a la presión, temperatura, caudal y densidad: valores altos (A), medios (M) o bajos (B); por otra parte, la turbidez y la ebullición son variables categóricas —ie., el contenido de la cuba se enturbia o no, hierve o no— por lo que se representarán por los valores sí (S) o no (N). En resumen, el dominio del problema consta de seis atributos con los siguientes valores:

Atributo		**Valor del atributo**	
P	Presión	$A/M/B$	Alta/Media/Baja
T	Temperatura	$A/M/B$	Alta/Media/Baja
$U?$	Se enturbia?	S/N	Sí/No
Q	Caudal	$A/M/B$	Alto/Medio/Bajo
D	Densidad	$A/M/B$	Alta/Media/Baja
$V?$	Hierve?	S/N	Sí/No

A cada asignación de valores en ese dominio se le llama *suceso*; a cada una de las asignaciones de valores de cada atributo —lo que en los AGs se llamaban *alelos*— se le dice *selector*. Se admiten las asignaciones múltiples (vg., $\langle P = \{M, A\}\rangle$) e indeterminadas (por defecto).

Es imposible en la práctica obtener leyes de control analíticas para ese problema; de hecho, ni siquiera es posible plantearlo como un problema de optimización, puesto que no hay ecuaciones fiables para la evolución temporal del modelo. Por eso se intenta el control a través de reglas de decisión. Con ellas se trata de describir conceptos o pautas de comportamiento ante ciertos sucesos.

Formalmente, cada concepto C es expresado en función de los atributos mediante una simplificación de la Lógica de Predicados denominada VL_1, de la siguiente manera:

$$c_1 \vee \ldots \vee c_r \Rightarrow C$$

donde los c_k son conjunciones de selectores. Un ejemplo lo aclarará: el concepto C_1 con el que se define una "fermentación excesivamente lenta" se describe así

$$(\langle P = A\rangle \wedge \langle D = A\rangle) \vee (\langle P = M\rangle \wedge \langle Q = M\rangle) \Rightarrow C_1$$

ie., la fermentación es lenta siempre que la presión y la densidad sean simultáneamente altas o bien la presión y el caudal simultáneamente intermedios. Otro ejemplo: se conviene en describir el concepto "fermentación con precipitación de posos" de la siguiente manera:

$$\langle U? = S\rangle \wedge \langle Q \neq A\rangle \Rightarrow C_2$$

Con esta terminología, el problema del aprendizaje inductivo mediante reglas consiste en construir un sistema lógico capaz de determinar si un suceso —posiblemente no identificado con anterioridad— está dentro o fuera de la del *campo semántico* (abreviadamente, *descripción*) de cierto concepto.

La formulación desarrollada por Holland y sus colaboradores en la Universidad de Michigan codifica las reglas a través de *clasificadores*. Un clasificador consta de dos partes, una correspondiente a un mensaje de condición y otra correspondiente a un mensaje de acción; la parte de condición es una cadena finita de valores sobre cierto alfabeto que incluye el símbolo '*' de indiferencia, la parte de acción es otra cadena finita sobre un alfabeto (habitualmente el mismo) que excluye el símbolo de indiferencia. En el presente ejemplo, cada clasificador tomará la siguiente forma:

$$(q_P\, q_T\, q_{U?}\, q_Q\, q_D\, q_{V?}) \;:\; d_1\, d_2$$

donde q_P pertenece al dominio de P, q_T pertenece al dominio de T, etc... y d_1 es una variable binaria con la que se indica que el suceso anterior es un *ejemplo positivo* del concepto C_1 y d_2 lo correspondiente.
Por ejemplo, el concepto C_1 se representa a través de estos dos clasificadores:

$$(\mathrm{A} * * * \mathrm{A} *) : 1\ 0$$
$$(\mathrm{M} * * \mathrm{M} * *) : 1\ 0$$

y el concepto C_2 a través de estos otros dos

$$(* * \mathrm{S}\ \mathrm{M} * *) : 0\ 1$$
$$(* * \mathrm{S}\ \mathrm{B} * *) : 0\ 1$$

Como ya se comentó, un SC mantiene una población de clasificadores interactuando con el entorno, la cual es actualizada a través de un AG con el fin de mejorar progresivamente la capacidad de desenvoltura del sistema en dicho entorno. Para medir esa 'capacidad de desenvoltura' se asocia a cada clasificador $\mathbf{q}_j$ una *fortaleza* s_j, la cual le servirá no sólo para participar en el AG, sino también para competir en la subasta con la que se determina qué clasificadores mandan sus mensajes.

Admítase, para simplificar el ejemplo, que el sistema debe aprender a discernir cuándo la fermentación transcurre demasiado lentamente, ie., el concepto C_1, y que se parte de la siguiente población de clasificadores:

$$\begin{array}{ll} \mathbf{q}_1 = (* * * \mathrm{A}\ \mathrm{A} *) : 1 & \quad s_1 = 12.3 \\ \mathbf{q}_2 = (* * \mathrm{S} * * \mathrm{M}) : 0 & \quad s_2 = 10.1 \\ \mathbf{q}_3 = (\mathrm{A}\ \mathrm{A} * * * *) : 1 & \quad s_3 = 8.7 \\ \mathbf{q}_4 = (* \mathrm{B} * * * *) : 0 & \quad s_4 = 2.3 \end{array}$$

Admítase también que desde el entorno llega el mensaje de un nuevo suceso, por ejemplo,

$$(\mathrm{A\ B\ S\ A\ A\ M})$$

que es una descripción de una cuba con la presión alta, baja temperatura, turbidez, alto caudal, alta densidad y sin ebullición. Obviamente, esa cuba constituye un ejemplo positivo del concepto C_1, por tener alta presión y alta densidad.

Ese mensaje activa tres clasificadores, $\mathbf{q}_1$, $\mathbf{q}_2$, y $\mathbf{q}_4$, los cuales deberán competir en la subasta por mandar sus respectivos mensajes de acción. Los clasificadores pujan proporcionalmente a su fortaleza, ie., $bid_j := b \cdot s_j$, pudiendo enviar su mensaje únicamente el clasificador más fuerte (esto es una convención; también podrían enviar sus mensajes los dos más fuertes), es decir $\mathbf{q}_1$. El 1 a la izquierda de los ':' indica que q_1 encaja en la descripción del concepto C_1, por lo que dicho clasificador obtendrá una recompensa $r > 0$ del entorno, de tal suerte que su fortaleza pasa a ser

$$s_1 \leftarrow s_1 - bid_1 + r$$

Si hubiera estado fuera de la descripción de C_1 la recompensa habría sido negativa.

El procedimiento anterior se reitera hasta que toque aplicar el AG. Entonces la fortaleza actual de cada clasificador se utilizará como medida de su aptitud. La selección se hace por sorteo y el reemplazo por factor de llenado, dado que no interesa tanto el mejor clasificador como una población completa que realice apropiadamente la labor de clasificación. En cuanto a los operadores genéticos, se usan los habituales de la representación entera, teniendo en cuenta únicamente que se conviene en que la fortaleza (aptitud) de un clasificador mutado se mantiene y que la fortaleza de la descendencia de un cruce es una media ponderada de las de sus progenitores. Una vez que ha terminado el AG, la población de clasificadores resultante sustituye a la anterior y se sigue con el proceso de adquisición de estímulos y emisión de respuestas. Por lo común, el AG se aplica una vez cada cierto número de pasos (T_{AG}), si bien en problemas particulares se utilizan criterios heurísticos.

Al final, la población de clasificadores debería converger a un conjunto de individuos de gran fortaleza con los que se describe lo mejor posible el concepto C_1.

Los sistemas clasificadores que se implantan para resolver 'problemas reales' son muy parecidos en lo fundamental al recién descrito. Quizá la diferencia más notable radique en el mecanismo de aportación de crédito, que no ha sido incluido en el presente ejemplo por no ser necesario.

El mecanismo de aportación de crédito se usa en aquéllas ocasiones en que la activación de una regla provoca un mensaje que a su vez activará otra regla, y así sucesivamente. Esta técnica de *encadenamiento de mensajes* proporciona gran flexibilidad al SC y permite emular un proceso de "razonamiento interno" (!) con el que adiquirir más fácilmente los conceptos; por ese motivo es muy utilizado en la práctica. Básicamente, el mecanismo de aportación de crédito no es más que un procedimiento de transmisión de recompensas con el que se reparte el crédito proporcionado por el entorno entre todos los clasificadores que han contribuído a la victoria de otro en la subasta. El procedimiento de transmisión de recompensas más común es el *algoritmo de relevos* (en bibliografía, *bucket brigade algorithm*), el cual se describe pormenorizadamente en [RIOL88] y [GOLD89,p225].

Además, la subasta suele incorporar un conjunto de procedimientos auxiliares de entre los que destacan:

- Un mecanismo con el que se introduce una componente aleatoria en el proceso de puja.
- Un sistema de penalizaciones o 'tasas' con el fin de evitar la convergencia prematura del sistema hacia las reglas más productivas.

En las referencias anteriormente señaladas se comentan e ilustran los detalles de la implantación de estos mecanismos.

En resumen, la implantación de un SC genérico responde a la siguiente estructura general:

```
BeginAlgorithm{LearningClassiffierSystem}
   MesList = InitML();      AuxMesList = InitML();
   P[0] = InitPop();        Q[0] = InitPop();
   Credit = InitCredit(P[0]);
   for( t = 0; t < MaxNumGen; t++) {
      MesBox = ReadDetectors();
      AuxMesBox = MatchClassifiers( MesBox, P[t] );
      AuxMesBox = SelectMatchingClassifiers( AuxMesBox, P[t] );
      AuxMesBox = TaxPostingClassifiers( AuxMesBox, P[t] );
      MesBox = WriteEffectors( AuxMesBox );
      Credit = ReceivePayoff( MesBox, t );
      Q[t] = DistributeCredit( Credit, P[t] );
      if( ApplyGA(t) )  then
         P[t] = UpdateRules( Q[t] );
      else
         P[t] = Q[t];
   }
EndAlgorithm
```

En ese esquema `MesBox` es la caja de mensajes internos y `AuxMesBox` una estructura auxiliar sobre la que se hacen operaciones. Análogamente `P[t]` es la población actual de reglas y `Q[t]` una población auxiliar sobre la que se harán operaciones. Tras las fases de inicialización, los detectores identifican los mensajes de entrada y los pasan a la caja de mensajes internos. A continuación se comparan los mensajes con los clasificadores (`MatchClassifiers()`) y se guardan las coincidencias en `AuxMesBox`. A través de una subasta (`SelectMatchingClassifiers()`), los clasificadores más fuertes de `AuxMesBox` envían sus mensajes a los actuadores y pagan por ello, reduciendo su fuerza. Después se recibe el refuerzo del entorno y con él se actualizan los créditos distribuyéndolo entre los clasificadores de la población actual. Por último, cada cierto número de iteraciones se hace evolucionar a la población de clasificadores (`ApplyGA()`) con el fin de tener una población de reglas cada vez mejor.

7.1.2 Formulación alternativa (de Pittsburg) del problema de aprendizaje evolutivo basado en reglas

Si se utiliza la formulación tradicional para implantar los sistemas clasificadores debe tenerse especial cuidado en diseñar los mecanismos de subasta y aportación de crédito para evitar que el SC se ofusque y pueda efectivamente extraer conclusiones de los eventos que se le presentan.

Con el fin de obviar estos y otros problemas se ha desarrollado en la Universidad de Pittsburg un enfoque alternativo que concede más importancia al proceso evolutivo del SC en detrimiento del proceso cognitivo. Se trata ahora de utilizar unas estructuras de datos más ricas y unos operadores más flexibles para hacer competir poblaciones enteras de reglas de decisión —mejor que reglas solas— y obtener finalmente la población de reglas que más fielmente cubren un conjunto de conceptos. Nótese que ahora el algoritmo evolutivo no es un mecanismo del proceso de aprendizaje, sino que el propio propio proceso de aprendizaje se ha reducido a un procedimiento evolutivo.

Este planteamiento es mucho más próximo a la idea de la computación evolutiva; sin embargo, se ha usado menos que el de Michigan debido a las dificultades para encontrar una representación y unos operadores lo suficientemente flexibles como para poder codificar eficientemente reglas complicadas. Los recientes avances en la representación de Sistemas de Aprendizaje Inductivo por un lado y el desarrollo de los Programas de Evolución (PEs, ver capítulo 5) por el otro han permitido desarrollar modelos prácticos de sistemas evolutivos de aprendizaje por inducción que reflejan el planteamiento de Pittsburg.

En la formulación de Pittsburg en cada individuo se codifica una solución completa del problema de adquisición inductiva de conceptos; por eso, cada uno contiene una disyunción de cierta cantidad variable de complejos, así:

$$\langle \underbrace{(s_{11}|\ldots|s_{1k})}_{c_1} \vee \underbrace{(s_{21}|\ldots|s_{2m})}_{c_2} \vee \ldots \quad \ldots \vee \underbrace{(s_{r1}|\ldots|s_{rm})}_{c_r} \rangle \Rightarrow C$$

Cada complejo es una conjunción de una cantidad fija de selectores que corresponden a otros tantos atributos, como ya se sabe. Obsérvese que ahora la longitud de los individuos no es fija.

Cada selector se codifica como una cadena binaria que contiene tantos elementos como valores tenga el correspondiente atributo, de manera que un `1` indica 'valor activado' y una colección de todos `1` para un selector deja el atributo con una asignación indiferente.

Siguiendo con el ejemplo anterior, el concepto C_1, queda descrito por el individuo

$$\langle (A * * * A*) \vee (M * * M * *) \rangle \Rightarrow C_1$$

o en notación desplegada

$$\langle \overset{P}{\texttt{100}} \mid \overset{T}{\texttt{111}} \mid \overset{U?}{\texttt{11}} \mid \overset{Q}{\texttt{111}} \mid \overset{D}{\texttt{100}} \mid \overset{V?}{\texttt{11}} \vee \overset{P}{\texttt{010}} \mid \overset{T}{\texttt{111}} \mid \overset{U?}{\texttt{11}} \mid \overset{Q}{\texttt{010}} \mid \overset{D}{\texttt{111}} \mid \overset{V?}{\texttt{11}} \rangle \ : \ C_1$$

En ella, los bits de los selectores trivaluados (ie., P, T Q y D) valen, respectivamente, A, M y B y los de los selectores bivaluados (ie., U? y V?) valen respectivamente S y N. Se entiende que las asignaciones múltiples son disyunciones.

La formulación de Pittsburg trata resolver el problema de aprendizaje inductivo de un concepto haciendo evolucionar una población de individuos como el anterior hasta llegar a uno que cubra lo mejor posible el campo semántico del concepto en cuestión. Esa evolución se implanta mediante un programa de evolución (PE) como los del capítulo 5. Tal implantación no es sencilla, ya que se requiere una buena función de aptitud que mida el 'grado de cobertura' de cada individuo y un conjunto flexible, completo y consistente de operadores genéticos con significado para el problema.

En [MICH94,p279] se da noticia de un PE diseñado por C. Janikow que resuelve satisfactoriamente un problema muy similar al del control de la cuba de fermentación. Dicho PE dispone de operadores genéticos aplicables en tres niveles: nivel de individuo, de complejo y de selector. Un ejemplo de operador unitario en el nivel de individuos es *GeneralizaRegla*, el cual hace una conjunción lógica elemento a elemento de un subconjunto aleatorio de complejos en cierto individuo, es decir:

$$\langle c_1 \vee c_2 \vee c_3 \rangle \;\longrightarrow\; \langle c_1 \vee (c_2 \mathbin{\&\&} c_3) \rangle$$

Un ejemplo de operador binario en el nivel de individuo es *CopiaRegla*, el cual copia aleatoriamente complejos de un progenitor a otro, ie.:

$$\langle c_1 \vee c_2 \rangle \quad \langle c_3 \vee c_4 \rangle \;\longrightarrow\; \langle c_1 \vee c_3 \vee c_4 \rangle \quad \langle c_2 \rangle$$

Un ejemplo de operador en el nivel de complejos es *EliminaSelector*, por el que se fija al azar un selector de cierto complejo y lo elimina, es decir, pone todos los bits de su dominio a `1`. De modo parecido, los operadores en el nivel de selectores activan o desactivan algunos bits del dominio de cierto selector.

7.2 Otros sistemas evolutivos de aprendizaje artificial

Se van a describir brevemente otras metodologías que utilizan procedimientos propios de la computación evolutiva para resolver problemas de aprendizaje artificial. En realidad son metodologías lo suficientemente amplias y completas como para resolver cualquier tipo de problemas, pero a efectos prácticos su utilidad queda restringida a ciertos campos del aprendizaje artificial.

7.2.1 La Programación Genética

A principios de la década de los '90 J.R. Koza propuso utilizar la metodología de la computación evolutiva no para encontrar soluciones a los problemas, sino para encontrar el mejor procedimiento con que resolverlos. Dicho de otra manera, las estructuras que se someten a evolución no codifican las posibles soluciones a un problema sino algoritmos o programas que, al ser ejecutados, determinan dichas soluciones. A esta metodología se la conoce como *Programación Genética*.

El paradigma de los PEs resulta ser el más apropiado para llevar a cabo esta idea. La representación de los individuos se hace a través de unas estructuras de datos jerárquicamente enlazados que reciben el nombre de *árboles de análisis* (en la bibliografía, *parse trees*); la codificación utiliza un conjunto de símbolos de función y otro de símbolos terminales adecuados para la solución de problemas en el dominio de interés. Ambas cosas se pueden implantar muy cómodamente en lenguaje LISP usando las 'expresiones-S'. La función de aptitud mide la precisión de cada individuo al ejecutar una serie de ejemplos para los que la solución es conocida de antemano. En cuanto a los operadores genéticos se usan los clásicos, adaptados para trabajar eficientemente con árboles de análisis, y otros específicos de esta metodología.

La programación genética se utiliza principalmente para resolver problemas de aprendizaje artificial, aunque también ha dado buenos resultados resolviendo ciertos tipos de problemas combinatorios.

7.2.2 La Programación Evolutiva de Fogel

El paradigma de la Programación Evolutiva hace evolucionar pequeñas máquinas de estados finitos haciendo uso de dos operadores genéticos: la selección y la mutación. La mutación realiza cambios más o menos severos en la estructura de las máquinas de acuerdo con cierta distribución. La selección se hace por torneo o de modo determinista. No se exige que el tamaño de la población sea constante, tan sólo que cada individuo genere a lo sumo un descendiente para la próxima generación.

La programación evolutiva se usó para resolver problemas de reconocimiento de símbolos; tras un largo periodo en que su desarrollo pareció estancarse recientemente ha vuelto a ser considerada como un método eficiente para resolver ciertos tipos de problemas que puedan ser representados con naturalidad a través de máquinas de estados finitos.

Conclusiones

Los capítulos precedentes son el resultado de un detallado estudio de la Computación Evolutiva enfocado hacia el desarrollo aplicaciones en el ámbito empresarial. Para llevar a cabo dicho estudio ha sido necesario realizar una amplia búsqueda bibliográfica sobre varios libros específicamente dedicados a ese tema y más de setenta artículos especializados. También se ha realizado una minuciosa búsqueda en la red INTERNET de más de cuarenta herramientas informáticas, entre rutinas y aplicaciones, con intención de que sirvan de ayuda en la implantación de algoritmos evolutivos. Debido la gran dispersión de la información existente sobre esta materia, y dado que aún no han sido establecidos criterios concretos y consistentes para su aplicación en problemas reales, se ha prestado especial cuidado en la selección y ordenación de dicha información, con el fin de que pueda ser útil a futuros desarrolladores de algoritmos evolutivos.

Se han elaborado los siete capítulos con el propósito de constituir un texto de referencia que presente de modo coherente y pormenorizado los temas más relevantes de la Computación Evolutiva. En dicho texto se estudian los tres paradigmas más importantes de la Computación Evolutiva —a saber, los Algoritmos Genéticos, los Programas de Evolución y las Estrategias Evolutivas— enmarcándolos de manera natural en el contexto de los métodos de búsqueda y comparándolos en términos de generalidad y eficacia con los métodos clásicos de la disciplina.

Se ha dedicado la mayor atención a los Algoritmos Genéticos por constituir el paradigma más claro, flexible y general de toda la Computación Evolutiva, tanto a nivel teórico como práctico. Se ha desarrollado una formulación unificada para tratar con sencillez todas las variedades y extensiones existentes del modelo básico de Algoritmos Genéticos. De acuerdo con esa formulación un Algoritmo Genético consta de tres procedimientos básicos —selección, transformación y reemplazo—, fácilmente representables mediante diagramas de bloques. A su vez, la implantación de los tres procedimientos requiere la elección de nueve métodos y criterios, por lo que se estudian con detalle las diversas posiblidades de realizar tal elección. El análisis teórico de los Algoritmos Genéticos no sólo explica su eficiencia añadida en términos de paralelismo implícito, sino que también es la clave para entender las dificultades y limitaciones prácticas que plantea la búsqueda genética. Se analizan detenidamente las causas de tales dificultades y limitaciones y, con el fin de eliminarlas o reducirlas dentro de su propio ámbito, se proponen modificaciones y extensiones del modelo básico.

Sobre la base del exhaustivo análisis anterior se acomete el estudio de los paradigmas de los Programas de Evolución y las Estrategias Evolutivas. Éstos pueden ser vistos respectivamente como generalizaciones y extensiones del paradigma de los Algoritmos Genéticos, por lo que dicho estudio se centra en establecer claramente las peculiaridades propias de cada caso y en el desarrollo de ejemplos característicos.

Se ha dedicado un capítulo a la resolución de problemas de aprendizaje artificial sirviéndose de algoritmos evolutivos, enumerando todos los procedimientos necesarios para conseguir un sistema con aprendizaje e ilustrándolo sobre un ejemplo tomado de la industria.

Todos los algoritmos analizados se han concretado en otros tantos seudocódigos básicos de inmediata implantación mediante lenguajes de programación de alto nivel. Todos los seudocódigos han sido depurados y contrastados con otras rutinas de demostrada eficacia. Se pretende con ello reducir el tiempo de implantación de nuevas aplicaciones o de adaptación de las ya existentes.

Se ha resuelto un problema real de minimización de costes de seguimiento de procesos dinámicos utilizando Estrategias Evolutivas, el cual es difícilmente resoluble por métodos no evolutivos. Con ello se ha puesto de manifiesto la gran flexibilidad y generalidad de los Algoritmos Evolutivos, incluso en su faceta más específica.

En definitiva cabe concluir que, desde un punto de vista teórico, la principal innovación de los Algoritmos Evolutivos con respecto a otras técnicas de búsqueda es la introducción de mecanismos de selección de los candidatos a solución en función de sus respectivas aptitudes y de mecanismos de construcción de nuevos candidatos por recombinación y alteración ciega de los ya existentes. Unos y otros mecanismos proporcionan *robustez* a la búsqueda, esto es, le añaden eficiencia sin menoscabo de su generalidad.

A efectos prácticos, y más específicamente a la hora de desarrollar aplicaciones industriales, la robustez de los Algoritmos Evolutivos se hace patente en estas buenas propiedades:

- Por ser métodos de búsqueda ciega requieren información poco específica para funcionar; concretamente, basta con un criterio de aptitud que ni siquiera tiene que estar expresado explícitamente (puede ser un procedimiento), ni ser exacto (es insensible al ruido). A consecuencia de ello son procedimientos de búsqueda extremadamente generales.

- En muchos casos resulta sencillo incorporar información genérica; la añadidura de información genérica, por poca que sea, tiene efectos muy beneficiosos sobre su rendimiento sin perjudicar seriamente a su generalidad.

- Son métodos de aproximación sucesiva, pudiendo parar en cualquier momento para proporcionar buenas soluciones, no necesariamente óptimas.
- Trabajan con una *población* de posibles soluciones, lo que proporciona capacidad de elección de buenas soluciones en función de otros criterios menos precisos que el de la aptitud.
- Presentan una complejidad casi lineal, es decir el número de operaciones que realizan crece proporcionalmente con el tamaño del problema. Ello permite resolver aceptablemente problemas con gran número de variables utilizando una potencia computacional modesta.

Aunque el uso de Algoritmos Evolutivos no está exento de inconvenientes de índole técnico, se ha mostrado que dichos inconvenientes no son mayores que los planteados por otros métodos y que pueden resolverse sin mucho esfuerzo adicional. La única circunstancia en que los Algoritmos Evolutivos son netamente inferiores se da en la resolución de problemas que requieran una gran cantidad de conocimiento específico, motivo por el cual se desaconseja su utilización en favor de técnicas heurísticas.

En resumen, los Algoritmos Evolutivos ofrecen una combinación de simplicidad, flexibilidad y rapidez que les permite competir favorablemente con otros procedimientos generales de búsqueda.

Apéndice A

El programa GENESIS de J.J. Grefenstette (ver [GREF90]) devuelve varios archivos que contienen los resultados del problema resuelto así como ciertos datos y estadísticas de la evolución del Algoritmo Genético. Concretamente, para el problema del apartado 2.2.1 se introdujo la función objetivo en el archivo `in.riolo` así

```
/********************************************* archivo f0.c *************/

double eval(char str[], int length, double vect[], int genes)
{
   int i;
   double sum;

   sum = 0.0;
   for( i=0 ; i < genes ; i++ )
      sum += vect[i]  + fabs(32 * M_PI * vect[i]);

   return(sum);
}

/******************************************** fin del archivo ************/
```

Tras una adecuada parametrización el programa fue ejecutado, proporcionando en el archivo `rep.riolo` un informe de su evolución

```
rep.riolo for ga.f0
Sun Dec  3 18:55:35  1995

      Experiments = 1
     Total Trials = 1500
  Population Size = 30
 Structure Length = 12
   Crossover Rate = 0.600
    Mutation Rate = 0.001
   Generation Gap = 1.000
   Scaling Window = 5
  Report Interval = 30
 Structures Saved = 10
Max Gens w/o Eval = 2
```

```
   Dump Interval = 0
     Dumps Saved = 0
         Options = acfLM
     Random Seed = 123456789
        Rank Min = 0.750
```

MEAN

Gens	Trials	Lost	Conv	Bias	Online	Offline	Best	Average
0	30	0	0	0.575	1.278e+00	1.816e+00	1.962e+00	1.278e+00
1	60	0	0	0.564	1.343e+00	1.889e+00	1.962e+00	1.407e+00
2	90	0	0	0.600	1.389e+00	1.914e+00	1.962e+00	1.480e+00
3	120	0	0	0.600	1.412e+00	1.929e+00	1.979e+00	1.484e+00
4	150	0	0	0.633	1.454e+00	1.939e+00	1.979e+00	1.623e+00
5	180	0	0	0.633	1.480e+00	1.945e+00	1.979e+00	1.610e+00
6	210	0	0	0.656	1.505e+00	1.950e+00	1.979e+00	1.654e+00
7	240	0	0	0.658	1.532e+00	1.954e+00	1.979e+00	1.715e+00
8	270	0	1	0.675	1.557e+00	1.956e+00	1.979e+00	1.758e+00
9	300	1	2	0.706	1.576e+00	1.959e+00	1.979e+00	1.747e+00
10	330	2	2	0.708	1.596e+00	1.960e+00	1.979e+00	1.794e+00
11	360	2	2	0.706	1.613e+00	1.962e+00	1.979e+00	1.803e+00
12	390	1	2	0.700	1.627e+00	1.963e+00	1.979e+00	1.796e+00
13	420	1	2	0.703	1.641e+00	1.964e+00	1.979e+00	1.831e+00
14	450	2	2	0.725	1.657e+00	1.965e+00	1.979e+00	1.868e+00
15	480	2	2	0.733	1.670e+00	1.966e+00	1.979e+00	1.867e+00
16	510	2	2	0.739	1.682e+00	1.967e+00	1.979e+00	1.872e+00
17	540	2	2	0.756	1.691e+00	1.968e+00	1.984e+00	1.847e+00
18	570	2	2	0.753	1.700e+00	1.969e+00	1.984e+00	1.867e+00
19	600	2	2	0.739	1.710e+00	1.969e+00	1.984e+00	1.901e+00
20	630	2	2	0.739	1.719e+00	1.970e+00	1.984e+00	1.899e+00
21	660	2	2	0.783	1.726e+00	1.971e+00	1.984e+00	1.878e+00
22	690	2	2	0.789	1.735e+00	1.971e+00	1.984e+00	1.924e+00
23	720	2	2	0.789	1.741e+00	1.972e+00	1.984e+00	1.882e+00
24	750	2	3	0.789	1.746e+00	1.972e+00	1.984e+00	1.853e+00
25	780	2	3	0.783	1.751e+00	1.973e+00	1.984e+00	1.895e+00
26	810	2	3	0.781	1.756e+00	1.973e+00	1.984e+00	1.888e+00
27	840	2	3	0.781	1.762e+00	1.973e+00	1.984e+00	1.917e+00
28	870	2	3	0.786	1.767e+00	1.974e+00	1.984e+00	1.899e+00
29	900	2	3	0.797	1.771e+00	1.974e+00	1.984e+00	1.887e+00
30	930	2	3	0.797	1.774e+00	1.975e+00	1.984e+00	1.856e+00
31	960	2	3	0.792	1.775e+00	1.975e+00	1.984e+00	1.827e+00
32	990	2	3	0.797	1.778e+00	1.975e+00	1.984e+00	1.851e+00
33	1020	2	4	0.808	1.779e+00	1.975e+00	1.984e+00	1.817e+00
34	1050	2	4	0.817	1.781e+00	1.976e+00	1.984e+00	1.863e+00
35	1080	2	3	0.828	1.782e+00	1.976e+00	1.984e+00	1.826e+00
36	1110	2	3	0.842	1.785e+00	1.976e+00	1.984e+00	1.878e+00
37	1140	2	3	0.847	1.787e+00	1.976e+00	1.984e+00	1.854e+00
38	1170	2	4	0.861	1.790e+00	1.977e+00	1.984e+00	1.903e+00
39	1200	2	4	0.864	1.792e+00	1.977e+00	1.984e+00	1.866e+00
40	1230	2	5	0.867	1.795e+00	1.977e+00	1.984e+00	1.918e+00
41	1260	2	5	0.867	1.798e+00	1.977e+00	1.984e+00	1.918e+00
42	1290	2	5	0.867	1.798e+00	1.977e+00	1.984e+00	1.822e+00

43	1320	3	5	0.867	1.800e+00	1.977e+00	1.984e+00	1.877e+00
44	1350	3	6	0.875	1.802e+00	1.978e+00	1.984e+00	1.904e+00
45	1380	3	5	0.881	1.805e+00	1.978e+00	1.984e+00	1.926e+00
46	1410	3	5	0.878	1.808e+00	1.978e+00	1.984e+00	1.925e+00
47	1440	3	5	0.869	1.810e+00	1.978e+00	1.984e+00	1.921e+00
48	1470	4	5	0.869	1.812e+00	1.978e+00	1.984e+00	1.935e+00
49	1500	4	5	0.864	1.815e+00	1.978e+00	1.984e+00	1.946e+00

El archivo anterior muestra primeramente un informe sobre la parametrización realizada, después proporciona un resumen de cómo ha evolucionado la población. Son especialmente interesantes las columnas 5ª, 8ª y 9ª en las que se da cuenta de las evoluciones de la diversidad, la aptitud máxima y la aptitud media, respectivamente.

La última generación se muestra al completo en el archivo `ckpt.riolo`. Se ha usado precisamente ese archivo para dibujar la gráfica 7.2 en la que se muestra el estado de dicha generación de forma más visual:

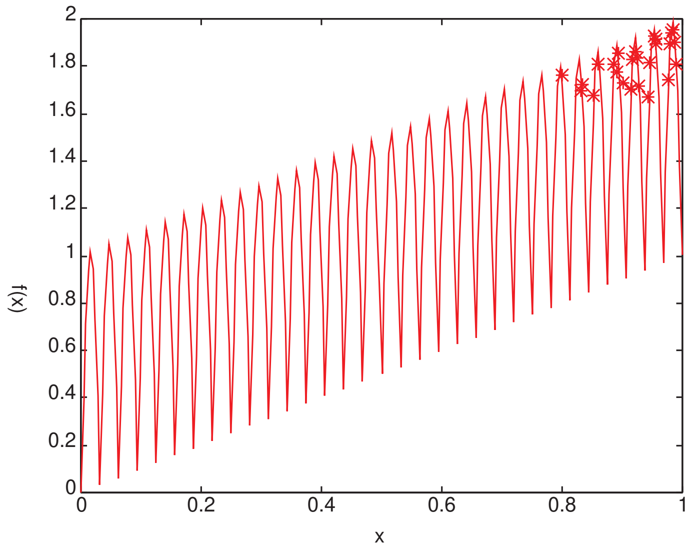

Figura 7.2: Última generación

Apéndice B

En este apéndice se muestra el listado del programa `ESTrack`, que ha sido usado para resolver el problema de seguimiento del apartado 6.4. Está escrito en el lenguaje de alto nivel de SCILAB salvo las rutinas gráficas, que en el original eran llamadas al programa de representaciones gráficas GNUPLOT y aquí se han sustituido por comandos gráficos de MATLAB para mayor sencillez y compatibilidad.

```
function J = ESTrack(Times,MaxIter,adap)
% ESTrack(Times, MaxIter, adap) realiza ´MaxIter´ iteraciones ´Times´ veces
%    para resolver el ´problema de la minimizacion de los costes de
%    seguimiento de procesos dinamicos´ a trav\´es de una Estrategia Evolutiva
%    Simple [MICH94,p169]. Se recomienda usar Times > 15 y Maxiter > 50.
%    ´adap´ es la tasa de adaptacion de las dispersiones; por defecto toma el
%    valor adap = 1.22 y se recomienda usar valores similares.


if nargin ~= 3, adap = 1.22; end;

num = 2;   den = [1 3 2 0];
Ts = 0.05;
[dNum, dDen] = c2dm(num, den, Ts, ´zoh´);
   % La planta se obtiene por discretizacion de una continua

N = 150;          % Numero de muestras (k=1,...,N) a simular

weights = ones(1,N);  %  esto es, C_al[k] = C_op[k] = 1


v_ini = [30, -60, 25, 1 -1];    % valores inciales de los parametros


vmin = [-200 -200 -200 -200 -200];    vmax = [200 200 200 200 200];
clc;
for j = 1:Times,
   v = v_ini;   stdev = 1;        % ie., valores iniciales para ´v´ y ´sigma´

   num_ctrl = v(1:3);             % Se determina un controlador
   den_ctrl = [1 v(4) v(5)];      %  de segundo orden
```

```
[num_ol, den_ol] = series(num_ctrl, den_ctrl, dNum, dDen);
[num, den] = cloop(num_ol, den_ol, -1);

y = dstep(num,den,N);          % respuesta dinamica a un escalon
y_ss = ddcgain(num,den);       % respuesta estacionaria a un escalon
e = abs(y-y_ss);               % %%% FUNCION DE ERROR %%% %

FObj = weights*e;              % Costes totales
ff = FObj;

foblog = zeros(1,MaxIter);     % En estos vectores se almacena la
suclog = zeros(1,MaxIter);     %  evolucion de las variables mas
devlog = zeros(1,MaxIter);     %  significativas

CntSuccess = 0;   CntFail = 0;
SuccessRate = 0;

for i = 1:MaxIter              % BUCLE PRINCIPAL DE LA (1+1)ES
   home;
   vv = v + randn(1,5) / stdev;

   num_ctrl = vv(1:3);
   den_ctrl = [1 vv(4) vv(5)];

   [num_ol, den_ol] = series(num_ctrl, den_ctrl, dNum, dDen);
   [num, den] = cloop(num_ol, den_ol, -1);

   y = dstep(num,den,N);
   y_ss = ddcgain(num,den);
   e = abs(y-y_ss);

   f = weights*e;

   TestFObj = (f < FObj);
   TestUBounds = all(vmax - vv > 0);          % Comprobacion de la
   TestLBounds = all(vv - vmin > 0);          %  factibilidad
   % TestConstraints = 1;
   if TestFObj & TestUBounds & TestLBounds,   % Mutacion con exito
      v = vv;
      FObj = f;
      CntSuccess = CntSuccess + 1;
      CntFail = 0;
   else,
      CntFail = CntFail + 1;
      if CntFail == 25,            % Se aborta la repeticion actual
         break;                    %  si la ES se estanca.
      end;
   end;
   SuccessRate = CntSuccess / i;
```

```
      if rem(i,3) == 0,            % Adapta las dispersiones cada tres pasos
         if SuccessRate < 0.2,
            stdev = stdev * adap;
         elseif SuccessRate > 0.2,
            stdev = stdev / adap;
         end;
      end;

      foblog(i) = FObj;
      suclog(i) = SuccessRate;
      devlog(i) = stdev;

      disp(['Resultados del paso ' int2str(j) ':' int2str(i)]);
      disp(['     a_0       '  num2str(v(1))])
      disp(['     a_1       '  num2str(v(2))])
      disp(['     a_2       '  num2str(v(3))])
      disp(['     b_1       '  num2str(v(4))])
      disp(['     b_2       '  num2str(v(5))])
      disp(['    COSTES     '  num2str(FObj)]);
      disp(['  Succ. Rate   '  num2str(SuccessRate)]);
      disp(['   Std. Dev.   '  num2str(stdev)]);
   end;

   if FObj < ff,
      ff = FObj;
      v_ini = v;
   end;

end;

num_ctrl = [ v(1)  v(2)  v(3)];
den_ctrl = [1   v(4)   v(5)];

[num_ol, den_ol] = series(num_ctrl, den_ctrl, dNum, dDen);
[numm, denn] = cloop(num_ol, den_ol, -1);

y = dstep(numm,denn,N);

clg;
stairs(y);
pause;

J = FObj;
```

Se ha programado la rutina `ESTrack` para que sea fácilmente ampliable. En el capítulo 6 se dan algunas sugerencias.

Glosario

Seguidamente se muestra una tabla de equivalencias léxicas entre los términos utilizados en la presente memoria y los correspondientes términos de la bibliografía inglesa.

Acantilados de Hamming	Hamming cliffs
Acoplamiento	Coupling
Actuadores	Effectors
Agregación	Aggregation
Ajuste fino	Fine tunning
Alelo	Alelle
Algoritmo de relevos	Bucket brigade algorithm
Algoritmo codicioso	Greedy algorithm
Algoritmo evolutivo	Evolutionary algorithm
Algoritmo genético	Genetic algorithm
Algoritmo genético tramado	Messy genetic algorithm
Algortimo genético supervisor	Supervisor genetic algorithm
Alteración	Altering
Aprendizaje artificial	Machine learning
Aptitud	Fitness
Aptitud bruta	Raw fitness
Autosintonizado	Self tunning
Bloque constructivo	Building block
Búsqueda aleatoria	Random search
Búsqueda ciega	Blind search
Caja de mensajes internos	Internal message box
Carga genética	Genetic load
Clasificador	Classifier
Claves aleatorias	Random keys
Codificación	Coding
Codificación ordinal	Order-based coding
Computación evolutiva	Evolutionary computation
Computación natural	Natural computation

Convergencia prematura	Premature convergence
Conocimiento específico	Problem-specific knowledge
Criadores	Breeders
Cromosoma	Chromosome
Cruce	Crossover
Cruce por emparejamiento parcial	Partial matching crossover
Cruce por recombinación de rutas	Edge recombination crossover
Debilidad	Weakness
Deriva genética	Genetic drift
Descendencia	Offspring
Desorientación	Deception
Desplazamiento	Shifting
Detectores	Detector
Diploide	Diploid
Dominante	Dominant
Duración	Lifetime
Ecuación de crecimiento reproductivo	Reproductive schema growth eq.
Edad	Age
Eficiencia	Efficiency
Eficacia	Efectiveness
Emparejamiento restrictivo	Restrictive mating
Emulación	Emulation
Epistasis	Epistasis
Escalado lineal	Linear scaling
Escalado ordinal	Order-based (ranking) scaling
Escalado potencial	Power law scaling
Especiación	Speciation
Especie	Species
Esquema aventajado	Above-average schema
Esquema retrasado	Below-average schema
Estrategia evolutiva	Evolution strategie
Evalución	Evaluation
Exploración	Exploration
Explotación	Exploitation
Extravío	Misleading
Fenotipo	Phenotype
Filtrado	Filtering
Fortaleza	Strength
Función de compartición	Sharing function
Gen	Gene
Genotipo	Genotype
Hibridación	Hibridation

Individuo	Individual
Inversión	Inversion
Ligadura	Binding
Longitud característica	Defining length
Mecanismo de aportación de crédito	Apportionment of credit mech.
Mensaje de entrada	Input message
Mensaje de salida	Output message
Métodos de escalada	Hill-climbing methods
Migración	Migration
Modelo centralizado	Centralized model
Modelo distribuido	Distributed model
Modelo en red	Network model
Muestreo aleatorio simple	Simple random sampling
Muestreo conservador	Preservative sampling
Muestreo directo	Direct sampling
Muestreo excluyente	Extinctive sampling
Muestreo por restos	Remainder sampling
Muestreo por ruleta	Roulette wheel sampling
Muestreo por sorteo	Lottery sampling
Muestreo por torneo	Tournament sampling
Muestreo por valor esperado	Expected-value sampling
Mutación	Mutation
Opacidad	Blindness
Operador de rango dinámico	Dynamic range operator
Optimización de cajas negras	Black-box optimization
Parlelismo implícito	Implicit paralelism
Patrón de similitud	Similarity pattern
Penalización	Penalty
Planifiación de horarios	Timetable scheduling
Planificación de tareas	Job shop scheduling
Población auxiliar	Gene-pool
Posici-ón/ones	Loc-us/i
Presión selectiva	Selective pressure
Problema del viajante de comercio	Traveling Salesperson Problem
Progenitores	Parents
Programa de evolución	Evolution program
Programación evolutiva	Evolutive programming
Programación genética	Genetic programming
Prueba de contratividad	Contractive mapping test
Punto factible	Feasible point
Puntuación	Score
Recesivo	Abeyant
Recombinación	Recombination

Reemplazo	Replacement
Reemplazo con factor de llenado	Crowding factor replacement
Reemplazo inmediato	On-the-fly replacement
Regla de decisión	Decision rule
Representación	Representation
Representación de recorrido	Path representation
Representación ordinal	Ordinal representation
Reproducción	Reproduction
Restricción	Constraint
Robustez	Robustness
Segregación	Segregation
Selección	Selection
Selección de estado estacionario	Steady-state selection
Selector	Selector
Símbolo de indiferencia	Don't care symbol
Simulación	Simulation
Solución no dominada	Non-dominated solution
Subasta	Auction
Suceso	Event
Superindividuo	Superindividual
Tuncado sigma	Sigma truncation

Referencias bibliográficas

En esta sección se comentan brevemente los textos que han sido consultados en la elaboración de la presente memoria de proyecto. La mayoría de ellos son fácilmente accesibles, ya sea a través de las bibliotecas universitarias, ya en librerías especializadas o a través de la red INTERNET.

Los siguientes libros son referencias básicas imprescindibles para cualquiera que desee tener un conocimiento cabal de la Computación Evolutiva:

[GOLD89] David E. Goldberg. *Genetic Algorithms in Search, Optimization & Machine Learning*. Addison–Wesley Co., Inc, Reading, MA, 1989.

[MICH94] Zbignew Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer–Verlag, 1994.

[DAVI91] Lawrence Davis. *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, New York, 1991.

[HEIT96] J. Heitkotter & D. Beasley. *The Hitch–Hiker's Guide to Evolutionary Computation*. January 1996.

[PROC89] J. David Schaffer (ed.) *Proceedings of the Third International Conference on Genetic Algorithms*. Morgan Kaufmann Publishers, 1989.

Para quien simplemente desee una introducción al tema son recomendables estas otras:

[HOLL92] John H. Holland. *Algoritmos Genéticos*. Investigación y Ciencia, Septiembre 1992.

[RIOL92] Rick L. Riolo. *Los bits mejor dotados*. Investigación y Ciencia. Septiembre 1992.

[GOLD94] David E. Goldberg. *First Flights at Genetic-Algorithm Kitty Hawk*. IlliGAL Report No. 94008. October 1994.

Siguen siendo de nivel introductorio, pero con mayor extensión, las siguientes referencias:

[HERR94] F. Herrera et al. *Algoritmos Genéticos: Fundamentos, Extensiones y Aplicaciones*. Technical Report #DECSAI 94105, Universidad de Granada. Abril 1994.

[BEAS93] David Beasley et al. *An Overview of Genetic Algorithms: Fundamentals (Part I) and Research Topics (Part II)*. University Computing Nr 15. 1993.

Una introducción más general a los nuevos temas de la matemática aplicada —de entre los que destacan las teorías del caos, de catástrofes y, por supuesto, de sistemas evolutivos— tratados todos ellos con amenidad pero con inusual rigor, puede encontrarse en:

[HAKE86] Hermann Haken. *Fórmulas del éxito en la naturaleza*. Salvat. 1986.

Las siguientes referencias tratan temas generales directamente relacionados con la Computación Evolutiva o con alguno de sus paradigmas:

[BALU95] Shumeet Baluja (`baluja@cs.cmu.edu`). *An Empirical Comparison of Seven Iterative and Evolutionary Function Optimization Heuristics*. 1995.

[RIBE92] Jose Ribeiro Filho. *Genetic Algorithm Programming Environments*. IEEE Computer Journal, 1992.

Estas otras se refieren a temas más específicos que de modo más o menos amplio han sido tratados en la memoria:

[RIOL88] Rick L. Riolo. *CFS-C: A Package of Domain Independent Subroutines for Implementing Classifier Systems in Arbitrary, User-Defined Environments*. 1988.

[WILC95] Jason R. Wilcox. *Organizational Learning within a Learning Classifier System*. Tesis doctoral presentada en la Universidad de Illinois. 1995.

[KARG94] Hillol Kargupta & David E. Goldberg. *Decision Making in Genetic Algorithms: A Signal-to-noise Perspective*. IlliGAL Report No. 94004. August 1994.

[GOLD93] David E. Goldberg et al. *Rapid, Accurate Optimization of Difficult Problems Using Fast Messy Genetic Algorithms*. IlliGAL Report No. 93004. February 1993.

[SPRA93] Joachim Sprave (`sprave@ls11.informatik.uni-dortmund.de`). *Linear Neighborhood Evolution Strategy.*

Las siguientes referencias son sobre temas generales y específicos, no directamente relacionados con la Computación Evolutiva, que se tratan en lugares específicos de la memoria:

[PHIL89] Charles L. Phillips y H. Troy Nagle, Jr. *Sistemas de Control Digital. Análisis y diseño.* Gustavo Gili. 1989.

[HILL82] Frederick Hillier & Gerarld J. Lieberman. *Introducción a la investigación de operaciones.* McGraw Hill. 1982.

[BAZA93] M. S. Bazaraa et al. *Nonlinear Programming. Theory and Algorithms.* Wiley and Co. 1993.

[KINC93] D. Kincaid & W. Cheney. *Numerical Analysis.* Brooks-Cole. 1993.

[NILS89] Nils J. Nilsson. *Principios de Inteligencia Artificial.* Addison & Wesley. 1989.

Las referencias mostradas a continuación tratan sobre aplicaciones concretas de la Computación Evolutiva en diversos problemas del ámbito empresarial:

[BERN95] J.L. Bernal et al. *Diseño óptimo de redes de distribución de energía eléctrica utilizando algoritmos genéticos.* IBERDROLA, Secretaría de I+D. 1995.

[TATE95] David M. Tate & Alice E. Smith *Unequal-area facility layout by genetic search.* IIE Transactions Nr. 27. 1995.

[SCHU93] Alan C. Schultz et al. *Test and Evaluation by Genetic Algoritms* IEEE Transactions on Expert Systems, October 1993.

[UCKU93] Serdar Uckun et al. *Managing Genetic Search in Job Shop Scheduling* IEEE Transactions on Expert Systems, October 1993.

[JANS93] David J. Janson & James F. Fenzel. *Training Product Unit Neural Networks with Genetic Algorithms.* IEEE Transactions on Expert Systems, October 1993.

[BALL93] N. R. Ball et al. *Genetic algorithm representations for laminate layups.* Artificial Inteligence in Engineering Nr 8, 1993.

Durante la elaboración del proyecto se han recopilado más de cuarenta aplicaciones informáticas de dominio público que implantan diversos tipos de algoritmos evolutivos. Todas ellas están acaptablemente documentadas, destacando las siguientes por la calidad de su documentación:

[GREF90] John J. Grefenstette. A User's Guide to GENESIS. 1990.

[LEVI95] David Levine. *User's Guide to the PGAPack Parallel Genetic Algorithm Library*. 1995

[SGAC94] Robert E. Smith et al. *SGA-C: A C-Language Implementation of a Simple Genetic Algorithm*. The Clearinghouse for Genetic Algorithms, The University of Alabama, 1995.

[JANI93] Cezary Z. Janikow. *GenET User's Manual*. 1993.

[CANT94] Erick Cantú–Paz. *Distributed GENESIS User's Guide*. 1994.

[VOIG91] Hans-Michael Voigt et al. *The Evolution Machine Manual*. 1991.

[MERE94] J.J. Merelo. *GAGS User's Manual*. 1994.

[HUNT95] Andrew Hunter. *Sugal Programming Manual*. 1995.

[DEB93] Kalyanmoy Deb & David E. Goldberg. *mGAinC: A Messy Genetic Algorithm in C*. IlliGAL Report No. 91008. September 1991.

Naturalmente, la bibliografía recién descrita está muy lejos de ser completa; se han omitido algunas referencias fundamentales, bien porque no he tenido acceso a ellas, bien porque no las he considerado particularmente interesantes para el desarrollo de aplicaciones industriales. De todos modos, existen varias bibliografías generales sobre la Computación Evolutiva; las más completas son posiblemente estas dos:

[GOLD92] David E. Goldberg et al. *Genetic Algorithms: A Bibliography*. IlliGAL Report No. 92008. July 1992.

[BACK93] Thomas Back et al. *Applications of Evolutionary Algorithms, extended edition*. University of Dortmund. August 1993.

Créditos

Los manuales para la elaboración de trabajos de fin de carrera recomiendan no agradecer la colaboración de otras personas en dicho proceso. Lo justifican afirmando que un agradecimiento a alguien que está cumpliendo con su obligación —por ejemplo, los directores de proyecto— podría ser interpretado como una adulación, especialmente si la calificación del trabajo depende de ellos; por otra parte, el agradecimiento a alguien que no tiene obligación de colaborar en el trabajo entra dentro del ámbito estrictamente personal, y por ello es improcedente introducirlo en un documento técnico. No es que sea mi intención desoír los consejos de quienes saben mucho más que yo de estos menesteres, pero considero que es absolutamente inverosímil pretender que en la elaboración de un documento de casi doscientas páginas únicamente han intervenido el autor y sus directores. Por eso, aun dejando para otro momento la elemental cortesía, creo que es un acto de justicia el reconocer aquí la colaboración de las siguientes personas:

- Mª Dolores Serrada Bayón y Albino Pérez Pérez financiaron en su totalidad la realización del proyecto. Para ellos más que para nadie cabe decir que sin su cooperación este proyecto no habría podido llevarse a cabo.

- David Pérez Serrada me ayudó en la compilación de muchas de las rutinas que de dominio público que se utilizan en el proyecto, así como con otros muchos problemas de caracter informático que fueron surgiendo en todos estos meses. Su colaboración fue también fundamental en las fases finales de corrección e impresión de las copias definitivas.

- Roberto García García me ayudó con el programa de composición tipográfica LaTeX y leyó las primeras pruebas (las más desagradecidas) del presente proyecto, realizando juiciosos comentarios sobre lo que veía.

- José Carlos Martín Azurmendi me proporcionó apoyo informático para la elaboración de los dibujos. Asimismo hizo atinadas sugerencias sobre cuestiones concretas y generales de la memoria.

- La diligente doctora Anne Pick y su marido, el sagaz doctor Herb Pick, colaboraron a través de su hija, la diligente y sagaz violinista Cynthia Pick, proporcionándome imprescindibles referencias bibliográficas desde la lejana Universidad de Minnesota.

- Los profesores José Luis del Álamo Sarmiento, Enrique Baeyens Lázaro y Ioannis A. Dimitriadis me introdujeron en el campo de la Computación Evolutiva y me proporcionaron gran cantidad de referencias.

- Ricardo Sayalero González me dio provechosos consejos sobre el modo de enfocar esta memoria, así como sobre su presentación y defensa.

- Jesús Sáinz Estébanez y Fernando Lamas Ferreiro resolvieron de buen grado problemas con la terminología y el estilo.

- J.J. Merelo y F. Herrera resolvieron con prontitud y simpatía ciertas dudas que un desconocido les planteó a través del correo electrónico. Debo agradecer además a J.J. Merelo su amable disposición para revisar la versión *de dominio público* del presente documento, bien entendido que ello no le hace responsable de nada de lo que en él se dice.

- Multitud de usuarios del grupo de noticias `comp.ai.genetic` y de la lista de correo electrónico `gaphd.list` me fueron de gran ayuda en todas las fases de elaboración de ésta memoria al compartir sus dudas conmigo y responder a las que yo les planteé.

Vaya para todos ellos, ya que no mi agradecimiento, sí mi más sincero reconocimiento.