UNIVERSIDAD DE GRANADA

VISUALIZACIÓN DE ESPACIOS DE BÚSQUEDA MULTIDIMENSIONALES EN COMPUTACIÓN EVOLUTIVA

TESIS DOCTORAL

Gustavo Romero López

Directores:
Alberto Prieto Espinosa
Juan Julián Merelo Guervós

Granada, 2003

Departamento de Arquitectura y Tecnología de Computadores

UNIVERSIDAD DE GRANADA

VISUALIZACIÓN DE ESPACIOS DE BÚSQUEDA MULTIDIMENSIONALES EN COMPUTACIÓN EVOLUTIVA

Memoria presentada por

Gustavo Romero López

Para optar al grado de

DOCTOR EN INFORMÁTICA

Fdo. Gustavo Romero López

D. Alberto Prieto Espinosa, Catedrático de Universidad y **D. Juan Julián Merelo Guervós**, Profesor Titular de Universidad, del Departamento de Arquitectura y Tecnología de Computadores de la Universidad de Granada

CERTIFICAN

Que la memoria titulada: **VISUALIZACIÓN DE ESPACIOS DE BÚSQUEDA MULTIDIMENSIONALES EN COMPUTACIÓN EVOLUTIVA** ha sido realizada por **D. Gustavo Romero López** bajo nuestra dirección en el Departamento de Arquitectura y Tecnología de Computadores de la Universidad de Granada para optar al grado de Doctor en Informática.

Granada, a 17 de octubre de 2002

Fdo. Alberto Prieto Espinosa
Director de la Tesis

Fdo. Juan Julián Merelo Guervós
Director de la Tesis

A Elena

Agradecimientos

A Elena, por todo su cariño y compresión durante la redacción de este trabajo.

A mi familia, por todo lo que soy.

A mis directores de tesis, JJ y Alberto, por sus inestimables consejos e ideas acerca de la mejor forma de llevar a cabo esta memoria.

A Pedro, por sus valiosos consejos acerca de las redes neuronales.

A Maribel, por las muchas pequeñas cosas que me han ayudado a acabar el trabajo.

A Victor, por su forma de ver la vida.

También quisiera expresar mi agradecimiento a todos los que han participado de alguna forma en la realización de esta memoria, especialmente a mis compañeros del equipo GeNeura y del departamento.

Resumen

En esta tesis se propone un nuevo método de visualización multidimensional para algoritmos evolutivos basado en el empleo de los mapas autoorganizativos de Kohonen, y se demuestra su utilidad a través de la aplicación del mismo a un conjunto de problemas típicos del área.

El objetivo del método es poder representar en un plano el espacio de búsqueda de un algoritmo evolutivo independientemente de su número de dimensiones. La visualización puede ser útil en muchos aspectos, de entre los que cabe destacar la identificación de características de la población, la visualización del espacio explorado durante la ejecución de un algoritmo evolutivo y la visualización del espacio de búsqueda de un problema. Eventualmente, esto servirá para poder escoger tanto los operadores como los parámetros de un algoritmo evolutivo, y para comprobar si en efecto su funcionamiento es el deseado.

Abstract

In this thesis a new method of multidimensional visualization for evolutionary algorithms sets out based on the use of the Kohonen's Self-Organizing Maps, and shows its utility through the application to a set of typical problems of the area.

The objective of the method is to be able to represent in a plane the search space of an evolutionary algorithm problem independently of its number of dimensions. The visualization can be useful in many aspects as: identifying population features, visualizing the space searched during a evolutionary algorithm run and the visualization of a problem's entire search space. This will let us to choose operators and parameters for an evolutionary algorithm, and to verify if the evolution process is as desired.

Índice general

Índice de figuras

Capítulo 1

Introducción

En esta tesis se propone un nuevo método de visualización multidimensional para algoritmos evolutivos basado en el empleo de mapas autoorganizativos, y se demuestra su utilidad a través de su aplicación a un conjunto de problemas típicos del área.

El objetivo del método es poder representar en un plano el espacio de búsqueda de un algoritmo evolutivo independientemente de su número de dimensiones. Permite visualizar un espacio de búsqueda completo o ciertas partes del mismo, como, por ejemplo, las zonas exploradas durante su ejecución o por las que se desplazan los individuos de una población en un instante dado.

La visualización puede ser útil en muchos aspectos, de entre los cuales caben destacar la identificación de características de la población, la visualización del espacio explorado durante una ejecución de un algoritmo evolutivo y la visualización del espacio de búsqueda de un problema. Esto nos servirá para poder escoger tanto los operadores como los parámetros de un algoritmo evolutivo, y para comprobar como funciona y si en efecto evoluciona correctamente.

En este capítulo de introducción se muestra una motivación para el presente trabajo en la sección 1.1. En la sección 1.2, se realiza una breve revisión de los conceptos básicos acerca de los algoritmos evolutivos (AE) . A continuación, en la sección 1.3, se hace una somera descripción de los trabajos

existentes en cuanto a visualización de algoritmos evolutivos. Por último, en la sección 1.4, se describe la estructura general de este trabajo.

1.1. Motivación del presente trabajo

Para cualquier problema en que las soluciones ocupen un espacio de hasta tres dimensiones es relativamente sencillo poder dibujar el espacio de búsqueda del problema, y después, las soluciones sobre dicho espacio de búsqueda. En cambio, cuando el número de dimensiones es superior, puede ser difícil o imposible de representar. Para poder visualizar espacios de alta dimensionalidad será necesario algún método que nos permita pasar a un número de dimensiones adecuado a las capacidades de la visión humana. A estos métodos se les denomina de *proyección* o de *escalado dimensional*.

La intención de estos métodos es reducir no el número de datos sino el número de dimensiones de los mismos. El objetivo que se persigue es representar los datos originales, que poseen un elevado número de dimensiones, con un número de dimensiones menor pero conservando las mismas propiedades estadísticas. Las técnicas de proyección no solo reducen la complejidad del problema sino que también facilitan las tareas de visualización de los mismos al ser representados en un espacio de baja dimensionalidad.

Mediante los métodos de proyección podremos crea una representación del espacio de búsqueda de un problema multidimensional para así poder repetir cualquier proceso útil en el caso de problemas de menor dimensionalidad. En nuestro caso, nos centraremos en las tareas de visualización que permitan obtener información acerca del espacio de búsqueda de un problema o del área explorada por un algoritmo evolutivo durante su ejecución. Todo ello con el objetivo de poder obtener alguna información que nos permita mejorar nuestro conocimiento del problema y ayude a resolverlo mejor o más rápido.

1.2. Algoritmos evolutivos

La teoría de la evolución fue descrita por Charles Darwin (Darwin, 1859) veinte años después de su viaje por las islas Galápagos en el Beagle, en el libro *"Sobre el Origen de las Especies por medio de la Selección Natural"*. Este libro fue bastante polémico en su tiempo, y en cualquier caso es una descripción incompleta de la evolución. La hipótesis de Darwin, presentada junto con Wallace, que llegó a las mismas conclusiones independientemente, es que pequeños cambios heredables en los seres vivos y la selección natural son los dos hechos que provocan el cambio en la Naturaleza y la generación de nuevas especies. Pero Darwin desconocía cuál es la base de la herencia, pensaba que los rasgos de un ser vivo eran como un fluido, y que los "fluidos"de los dos padres se mezclaban en la descendencia; esta hipótesis tenía el problema de que al cabo de cierto tiempo, una población tendría los mismos rasgos intermedios, la denominada "catástrofe del gris".

Fue Mendel quien descubrió que los caracteres se heredaban de forma discreta, y que se tomaban del padre o de la madre, dependiendo de su carácter dominante o recesivo. A estos caracteres que podían tomar diferentes valores se les llamaron genes, y a los valores que podían tomar, alelos. En realidad, las teorías de Mendel, que trabajó en total aislamiento, se olvidaron y no se volvieron a redescubrir hasta principios del siglo XX. Además, hasta 1930 el geneticista inglés Robert Aylmer no relacionó ambas teorías, demostrando que los genes mendelianos eran los que proporcionaban el mecanismo necesario para la evolución.

Más o menos por la misma época, el biólogo alemán Walther Flemming describió los cromosomas como ciertos filamentos en los que se agregaba la cromatina del núcleo celular durante la división; poco más adelante se descubrió que las células de cada especie viviente tenían un número fijo y característico de cromosomas.

Y no fue hasta los años 50, cuando Watson y Crick descubrieron que la base molecular de los genes está en el ADN, ácido desoxirribonucleico. Los cromosomas están compuestos de ADN, y por tanto los genes están en los

cromosomas.

Al código genético se le llama genotipo y al cuerpo que construyen esas proteínas, modificado por la presión ambiental, la historia vital, y otros mecanismos dentro del cromosoma, se llama fenotipo.

Esta es la base de la teoría del neo-darwinismo, que afirma que la historia de la mayoría de la vida está causada por una serie de procesos que actúan en y dentro de las poblaciones: reproducción, mutación, competición y selección. La evolución se puede definir entonces como cambios en el conjunto genético de una población.

Sin embargo, la selección natural no es la única forma de selección que podemos encontrar en la Naturaleza. En diversas áreas, como la economía y los mercados, podemos ver cómo ciertas empresas evolucionan, "alimentándose" unas de otras, ocupando *"nichos económicos"*, tal y como describe Brian Arthur en (Arthur, 1990) y John Holland en (Holland, 1975). Así mismo existe una teoría sobre cómo evolucionan los "universos inflacionarios" (Linde, 1994), según la cual, universos con diferentes leyes físicas compiten entre sí, resultando ganadores algunos de ellos, ocupando así todo el espacio posible. Finalmente, incluso las ideas evolucionan (*evolución memética* – memetic evolution) (Heylighen, 1992), compitiendo por el espacio cerebral, reproduciéndose y mutando: una idea puede ser una canción, una cadena de texto o una imagen de televisión.

En todos estos ejemplos de evolución es difícil identificar un sustrato que evoluciona, pero sí encontramos muchas otras características que nos hacen pensar en ellos como en algún tipo de evolución.

1.2.1. Desarrollo de la informática evolutiva

Primera generación

Las primeras ideas para simular o imitar la evolución natural con el objeto de resolver problemas vinieron de Von Neumann, incluso antes del descubrimiento del ADN. Von Neumann afirmó que la vida debía de estar apoyada por un código que a la vez describiera como se puede construir un ser vivo, y tal

que ese ser creado fuera capaz de autorreproducirse; por tanto, un autómata o máquina autorreproductiva tendría que ser capaz, aparte de contener las instrucciones para hacerlo, de copiar tales instrucciones a su descendencia.

Sin embargo, no fue hasta mediados de los años cincuenta, cuando el rompecabezas de la evolución fue prácticamente completado, cuando Box comenzó a pensar en imitarla para, en su caso, mejorar procesos industriales. La técnica de Box (Box, 1957), denominada EVOP (Evolutionary Operation), consistía en elegir una serie de variables que regían un proceso industrial. Sobre esas variables se creaban pequeñas variaciones que formaban un hipercubo, variando el valor de las variables una cantidad fija. Se probaba entonces con cada una de las esquinas del hipercubo durante un tiempo, y al final del periodo de pruebas, un comité humano decidía sobre la calidad del resultado. Es decir, se estaba aplicando mutación y selección a los valores de las variables, con el objeto de mejorar la calidad del proceso. Este procedimiento se aplicó con éxito a algunas industrias químicas.

Un poco más adelante, en 1958, Friedberg y sus colaboradores pensaron en mejorar el funcionamiento de un programa usando técnicas evolutivas. Para ello diseñaron un código máquina de 14 bits, y cada programa tenía 64 instrucciones. Un programa llamado Herman, ejecutaba los programas creados, y otro programa, el Teacher o profesor, le mandaba a Herman ejecutar otros programas y ver si los programas ejecutados habían realizado su tarea o no. La tarea consistía en leer unas entradas, situadas en una posición de memoria, y debían depositar el resultado en otra posición de memoria, que era examinada al terminarse de ejecutar la última instrucción.

Para hacer evolucionar los programas, Friedberg hizo que en cada posición de memoria hubiera dos alternativas; para cambiar un programa, alternaba las dos instrucciones (que eran una especie de alelos), o bien reemplazaba una de las dos instrucciones con una totalmente aleatoria.

En realidad, lo que estaba haciendo es usar mutación para generar nuevos programas; al parecer, no tuvo más éxito que si hubiera buscado aleatoriamente un programa que hiciera la misma tarea. El problema es que la mutación sola, sin ayuda de la selección, hace que la búsqueda sea prácticamente

una búsqueda aleatoria.

Más o menos simultáneamente, Bremmerman (Bremermann, 1962) trató de usar la evolución para *"entender los procesos de pensamiento creativo y aprendizaje"*, y empezó a considerar la evolución como un proceso de aprendizaje. Para resolver un problema, codificaba las variables del problema en una cadena binaria de 0s y 1s, y sometía la cadena a mutación, cambiando un bit cada vez. Bremmerman trató de resolver problemas de minimización de funciones, aunque no está muy claro qué tipo de selección usó y el tamaño y tipo de la población. En todo caso, se llegaba a un punto, la "trampa de Bremmerman", en el cual la solución no mejoraba; en intentos sucesivos trató de añadir entrecruzamiento entre soluciones, pero tampoco obtuvo buenos resultados. Una vez más, el simple uso de operadores que creen diversidad no es suficiente para dirigir la búsqueda genética hacia la solución correcta; y ésto se logra aplicándolo a un concepto de la evolución darwiniano clásico: por mutación, se puede mejorar a un individuo; en realidad, la evolución actúa a nivel de población.

Segunda generación

El primer uso de procedimientos evolutivos en computación se debe a Reed, Toombs y Baricelli (Toombs & Barricelli, 1967), que trataron de hacer evolucionar un tahúr que jugaba a un juego de cartas simplificado. Las estrategias de juego consistían en una serie de 4 probabilidades de apuesta alta o baja con una mano alta o baja, con cuatro parámetros de mutación asociados. Se mantenía una población de 50 individuos, y aparte de la mutación, había intercambio de probabilidades entre dos padres. Es de suponer que los perdedores se eliminaban de la población (tirándolos por la borda). Aparte de, probablemente, crear buenas estrategias, llegaron a la conclusión de que el entrecruzamiento no aportaba mucho a la búsqueda.

Los ***algoritmos genéticos*** (AG) (Holland, 1975; Goldberg, 1989; Davis, 1991) fueron ideados por John Holland en los '60 y desarrollados e investigados por Holland y sus colegas de la Universidad de Michigan en los '60 y '70. En contraste con lo que otros investigadores intentaban, Holland no pre-

tendía resolver problemas específicos, sino estudiar formalmente el fenómeno de la adaptación tal y como ocurre en la naturaleza, y desarrollar métodos para aplicar los mecanismos de la adaptación natural a sistemas de computadores. Holland, en su libro *"Adaptation in natural and Artificial systems"* (Holland, 1975) presentó el algoritmo genético como una abstracción de la evolución biológica y dio los principios teóricos para la adaptación con algoritmos genéticos.

Más o menos a mediados de los años 60, Rechenberg (Rechenberg, 1965; Rechenberg, 1973) y Schwefel (Schwefel, 1975; Schwefel, 1977) describieron las ***estrategias de evolución*** (EE) (Schwefel, 1995; Bäck, 1996). Las estrategias de evolución son métodos paramétricos de optimización, que trabajan con poblaciones de cromosomas compuestos por números reales. Hay diversos tipos de estrategias de evolución, que se verán más adelante. En la más común, se crean nuevos individuos de la población añadiendo un vector mutación a los cromosomas existentes en la población; en cada generación, se elimina un porcentaje de la población, y los restantes generan la población total, mediante mutación y cruce. La magnitud del vector mutación se calcula adaptativamente. Una revisión sobre las estrategias evolutivas se puede encontrar en Bäck, Hoffmeister y Schwefel (Bäck *et al.* , 1991).

A partir de los años 60 se han desarrollado algoritmos o métodos que podríamos llamar *evolutivos modernos*, y se han seguido investigando hasta nuestros días. Algunos de ellos son simultáneos a los algoritmos genéticos, pero se desarrollaron independientemente sin conocimiento unos de otros. Uno de ellos, la ***programación evolutiva*** (PE) de Fogel, Owens y Walsh (Walsh, 1966; Fogel, 1995), se inició como un intento de usar la evolución para crear máquinas inteligentes, que pudieran prever su entorno y reaccionar adecuadamente a él. Para simular una máquina pensante, se utilizó un autómata celular. Un autómata celular es un conjunto de estados y reglas de transición entre ellos, de forma que, al recibir una entrada, cambia o no de estado y produce una salida.

Fogel trataba de hacer aprender a estos autómatas a encontrar regularidades en los símbolos que se le iban enviando. Como método de aprendizaje

usó un algoritmo evolutivo: una población de diferentes autómatas competía para hallar la mejor solución, es decir, predecir cual iba a ser el siguiente símbolo de la secuencia con un mínimo de errores; los peores 50 % eran eliminados cada generación, y sustituidos por otros autómatas resultantes de una mutación de los existentes.

De esta forma, se lograron hacer evolucionar autómatas que predecían algunos números primos (por ejemplo, uno de ellos, cuando se le daban los números más altos, respondía siempre que no era primo; la mayoría de los números mayores de 100 son no primos). En cualquier caso, estos primeros experimentos demostraron el potencial de la evolución como método de búsqueda de soluciones novedosas.

Finalmente, la ***programación genética*** (PG) (Koza, 1992) se basa en la evolución de programas. Normalmente la representación usada sigue la sintaxis de programas en LISP, que esencialmente son árboles cuyos nodos internos son operadores y los externos operandos. Este tipo de estructura de datos requiere el uso de operadores de mutación y cruce específicos.

Tercera generación

Michalewicz propuso en su libro *"Genetic Algorithms + Data Structures = Evolution Programs"* (Michalewicz, 1996) prescindir de la representación usual de los individuos en la población (en cadenas de bits o vectores de números reales, como se venía haciendo), y al mismo tiempo aplicar el paradigma principal de la programación procedural a la computación evolutiva: aplicar algoritmos a estructuras de datos, ya que son distintas y deben estar separadas, y formar programas evolutivos mediante la unión e interacción de ambos.

No obstante, la programación procedural no es el paradigma de programación más avanzado: hoy día, los lenguajes más utilizados son orientados a objetos. La computación evolutiva orientada a objetos podría definirse de la siguiente forma: *Algoritmos + estructuras de datos = Objetos Evolutivos* (OE). En la programación orientada a objetos, los algoritmos y las estructuras de datos a las que se aplican aquellos, se encuentran encapsulados en

clases, y la interacción con las instancias de esas clases se realiza a través de la interfaz de clase.

El nivel de abstracción que se consigue haciendo uso de Objetos Evolutivos es tal que, sin ser realmente algoritmos genéticos, ni programas evolutivos, ni programas genéticos, ni otros paradigmas evolutivos definidos previamente, cualquiera de estos paradigmas puede llegar a implementarse.

1.3. Visualización de algoritmos evolutivos

Implementar un algoritmo genético para resolver un problema puede ser relativamente sencillo, pero analizar los resultados para descubrir si el proceso es eficiente o no, o si puede ser mejorado, puede ser extremadamente difícil. Existe una multitud de opciones en cuanto a representación, elección de operadores y selección de parámetros que pueden hacer marcar una gran diferencia en cuanto a la velocidad y la calidad de los resultados. El hecho de que muchas de estas opciones no sean independientes hace que el análisis sea aún más difícil. Guardar toda la información producidad por dicho algoritmo suele producir una inmensa cantidad de datos que no pueden ser analizados convenientemente a mano. Se pueden seguir varios caminos para realizar dicho análisis:

1. Si hay suerte, las elecciones hechas cuando se configura el algoritmo serán lo suficientemente buenas como para no tener que reajustarlo, siempre y cuando los resultados que se obtengan cumplan con los requerimientos de calidad y tiempo de ejecución que se estimen oportunos.

2. Otro método puede ser seguir el conocimiento "a priori" sobre el problema en base a lo que han hecho otros o nuestra propia experiencia anterior. Así por ejemplo, se escogería una probabilidad de mutación igual a uno dividido por el número de genes, un operador de cruce estándar, es decir, confiaríamos en todas las soluciones que habitualmente suelen funcionar.

3. Otro método muy empleado es hacer un pequeño conjunto de pruebas en las que se va subiendo y bajando las tasas de aplicación de ciertos operadores para ver si el algoritmo mejora o empeora. Este método como ha estudiado Jones (Jones, 1995) no es suficientemente significativo.

4. Otra solución es hacer que el propio algoritmo vaya adaptando algunos de sus parámetros de forma automática durante su ejecución (Corne *et al.* , 1994; Tuson & Ross, 1998; Igel & Kreutz, 1999; Toussaint & Igel, 2002).

5. Utilizar algún método estadístico para descubrir la combinación ideal de parámetros y operadores, como, por ejemplo, el análisis de la varianza ANOVA (Castillo *et al.* , 2002; Rojas *et al.* , 2002).

Ninguna de las cuatro primeras soluciones es satisfactoria desde el punto de vista científico pues tratan cada ejecución del algoritmo evolutivo como una caja negra, y no se plantean cómo funciona en su interior. Hay un creciente interés en la visualización de algoritmos evolutivos (ECVW, 1999; Romero *et al.* , 2000; Romero *et al.* , 2001; Romero *et al.* , 2002). El método que aquí proponemos se basa en la observación de la evolución de la población sobre una representación del espacio de búsqueda creada mediante un mapa autoorganizativo. El quinto método, aunque más ortodoxo desde el punto de vista científico, tiene un coste computacional muy elevado dada la gran cantidad de experimentos que puede ser necesario realizar y además sigue sin aportar información acerca de la forma de funcionar del algoritmo. Sólo nos ayuda a escoger los valores de los parámetros de forma que su interacción sea óptima.

1.3.1. Estado del arte

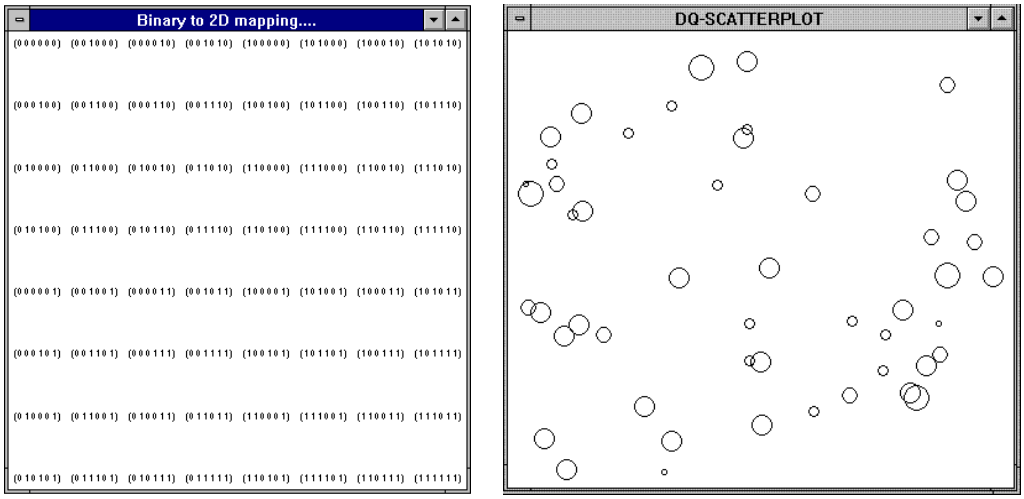
De acuerdo con Shine y Eick (Shine & Eick, 1997), la visualización de los algoritmos evolutivos puede ser útil para analizar en qué medida exploran sus espacios de búsqueda, analizar su convergencia, permitirnos obtener

algunas pistas sobre la forma de la función objetivo y la dinámica de su funcionamiento, y poder ajustar sus parámetros. Las tres primeras cuestiones han sido tratadas en específicamente en (Collins, 1997), (Collins, 1998b) y (Shine & Eick, 1997), y más recientemente en (ECVW, 1999).

Trevor Collins en (Collins, 1996) propone un método de representación denominado *"Genotypic-Space Mapping"*. Con él pretende poder representar poblaciones cuya representación sean grandes conjuntos de datos binarios, decimales, reales o de tipo cadena que posean un elevado número de dimensiones. Este método proporciona un mapeado lineal entre las cadenas originales de la población, en el espacio de alta dimensionalidad, a pares o ternas de coordenadas, de forma que la población pueda ser representada sobre un diagrama de dispersión. El método propuesto es analítico y en comparación con el algoritmo de Sammon obtienen tasas de error similares. En la figura 1.1a podemos encontrar un ejemplo de como se proyectan las 64 esquinas de un hipercubo binario de dimensión 6, $\mathbb{Z}_2^6$, sobre el plano, y en la figura 1.1b tenemos un ejemplo de como se proyecta una población utilizando este método.

Posteriormente, en (Collins, 1998b), intenta resolver el problema del ajuste de parámetros utilizando la visualización del espacio de búsqueda para permitir guiar al algoritmo de forma interactiva. Pero aunque este método puede tener éxito a la hora de encontrar buenos resultados, sigue sin mejorar nuestro conocimiento de por qué funciona el algoritmo. El principal punto flaco de este método es el gran error que suele obtenerse al utilizar este método de visualización. Este método no permite visualizar el espacio de búsqueda explorado por un algoritmo durante su funcionamiento, sino sólo la representación del todo el espacio de búsqueda del problema, es decir, es poco flexible a la hora de escoger como representar un espacio de búsqueda.

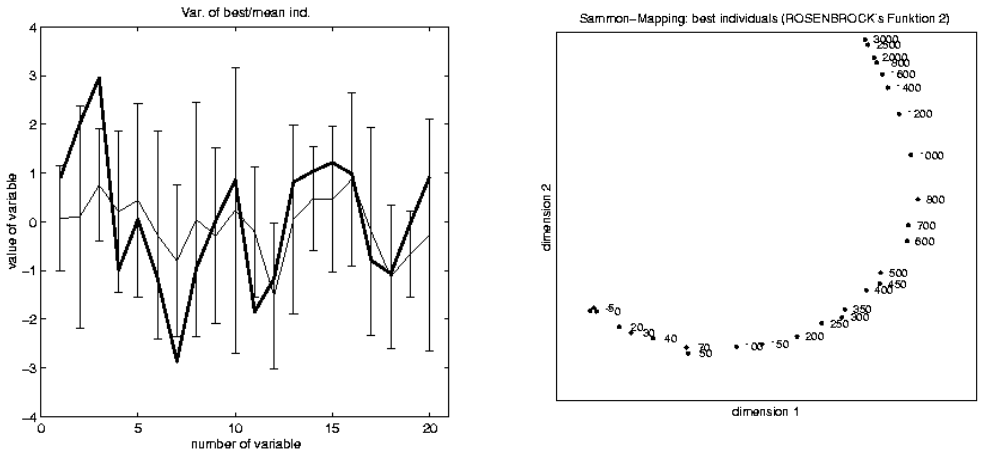
Hartmut Pohlheim en (Pohlheim, 1999) sugiere un conjunto de métodos de visualización para tipos de datos que a los que seguir la pista a lo largo de ciertos intervalos de tiempo para poder observar el curso y el estado de un algoritmo evolutivo. Estos tipos incluyen los genes y el valor de la función objetivo de los individuos, la distancia entre individuos y otras estadísticas



(a) Proyección de $\mathbb{Z}_2^6$ (b) Proyección de una población

Figura 1.1: Gráficos de Trevor Collins. (a) Mapeado de las 64 esquinas de un hipercubo binario de dimensión 6, $\mathbb{Z}_2^6$, sobre el plano. (b) Ejemplo de proyección de una población utilizando el método descrito por Trevor Collins. La posición indica como es el genotipo y el diámetro del circulo es mayor cuanto mejor es el individuo.

acerca de subpoblaciones que seguir a lo largo de periodos de tiempo que van desde una generación a varias ejecuciones. Proporciona ejemplos de algunos de las formas en que esto puede hacerse empleando gráficos bidimensionales, diagramas de barras e imágenes en color, todo ello implementado en Matlab (The MathWorks, Inc, 1994). Aunque los métodos propuestos por Pohlheim permiten una mejor comprensión de varios aspectos de un proceso evolutivo, en ningún caso permiten representar un espacio de búsqueda multidimensional. Algunos de los métodos que más se aproximarían a nuestro objetivo pueden los que se ven en la figura 1.2. En la figura 1.2a, la información que se da sobre cada individuo, salvo el mejor, es prácticamente nula pues sólo se aprecia la media y desviación típica del valor de cada variable de las que los componen. Para problemas multimodales o multiobjetivo la media puede no ser significativa, ya que puede existir diversos subóptimos. En el caso del método de visualización aplicado al la función de Rosenbrock (función banana) que podemos ver en la figura 1.2b, su principal inconveniente es que al

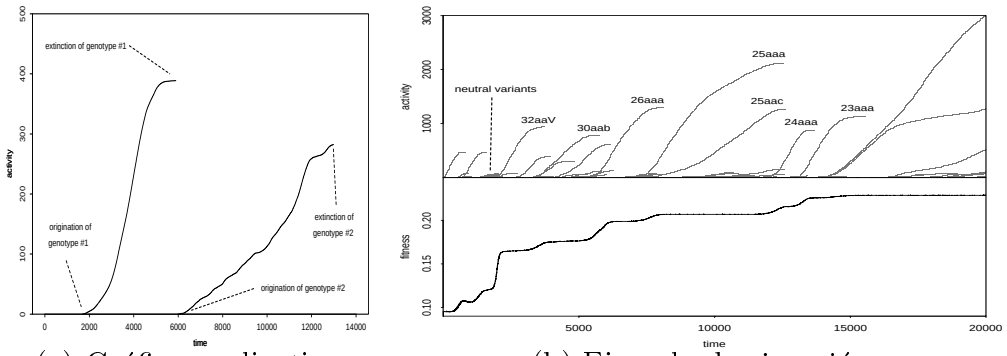


(a) (b)

Figura 1.2: Gráficos de Hartmut Pohlheim. (a) Valor medio y desviación de las variables de los individuos de una población. El mejor individuo está destacado en negrita (b) Función de Rosenbrock (función banana). Proyección 2D de una población reduciedo el número de dimensiones del problema desde las 10 originales a tan solo 2 usando el algoritmo de Sammon.

utilizar el algoritmo de Sammon como método de proyección, el proceso no es repetible para nuevos individuos y debe volver a realizarse a cada generación de forma que la imagen obtenida puede ser completamente diferente y no ser comparable con la de las demás generaciones.

Mark A. Bedau estudia la actividad evolutiva a través de los cambios que tienen lugar en los genotipos. Para visualizar dicha actividad representa en un plano la actividad de los genotipos frente al tiempo. Los genotipos más importantes dibujan una linea ascendente, parecida a una ola. La forma de las olas permite conocer datos acerca de la evolución. Así, si aparece un nuevo genotipo se eleva una nueva ola. La pendiente de la ola variará de acuerdo con la concentración del genotipo en la población, aumentando la pendiente cuando aumenta la concentración, y al revés, disminuyendo al caer dicha concentración. En el caso extremo de que el genotipo desaparezca, la pendiente llegará a cero y la ola acabará. En la figura 1.3 podemos ver un gráfico de ejemplo. En sus trabajos (Bedau *et al.* , 1999; Bedau & Brown,



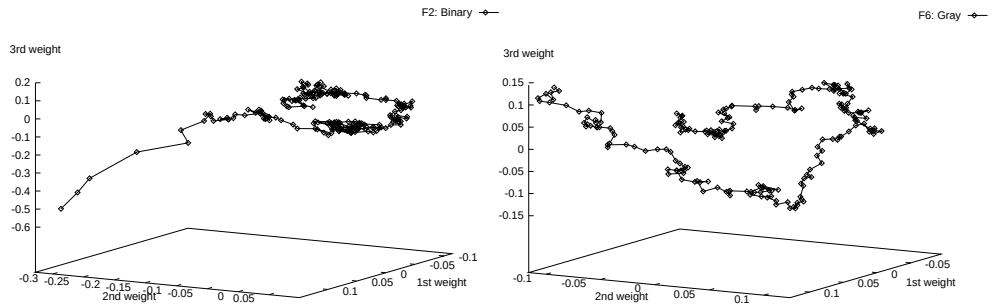
(a) Gráfico explicativo (b) Ejemplo de ejecución

Figura 1.3: Gráficos de Mark A. Bedau. (a) Explicación del método de visualización. (b) Ejemplo de la aplicación del método a una ejecución de un algoritmo evolutivo.

1999) utiliza **Evita**, un sencillo modelo de vida artificial que consiste en la evolución de programas en lenguaje ensamblador que se auto-reproducen y compiten por el espacio, de forma parecida a como sucede en **Tierra** (Ray, 1992). Este método que puede ser eficaz para seguir ciertos cambios en la estructura de las soluciones no es eficaz para aumentar nuestro conocimiento sobre el espacio de búsqueda ni sobre que áreas del mismo están siendo exploradas.

J.J. Collins emplea en (Collins, 1999) el Análisis de Componentes Principales como método de proyección con el objetivo poder extraer características significativas en las variaciones de los datos producidos por un algoritmo evolutivo. Se centra en el estudio de la eficiencia del proceso evolutivo y en descubrir si este ha quedado atrapado en un óptimo local. En la figura 1.4 tenemos las proyecciones 3D de dos funciones de De Jong, la función F2 utilizando codificación binaria y la función F6 (Rastrigin) empleando codificación de tipo Gray. Este método de proyección, al emplear el Análisis de Componentes Principales, tiene el mismo defecto que el algoritmo de Sammon, no pueden proyectarse nuevos individuos sin volver a ejecutar el algoritmo.

Annie S. Wu en (Wu *et al.* , 1999b; Wu *et al.* , 1999a) describe una herramienta, **VIS**, para analizar los datos producidos por en una o varias



(a) Función F2 de De Jong (b) Función F6 de De Jong

Figura 1.4: Gráficos de J.J. Collins. Camino recorrido por los mejores individuos obtenidos en la ejecución de un algoritmo genético proyectados en 3D empleando el Análisis de Componentes Principales: (a) Función F2 de De Jong (b) Función F6 de De Jong (Rastrigin).

ejecuciones de un algoritmo evolutivo. Permite mostrar las relaciones de parentesco entre individuos, así como identificar el origen y las posiciones de los genes en los que ha tenido lugar un cruce o mutación. Para ello almacena la historia completa de todas las generaciones de cromosomas, genes individuales y otros posibles tipos de esquemas. En la figura 1.5 podemos ver dos formas de visualización que este programa aporta, una de ellas proporciona una vista de una población, la otra de un ejecución completa. Esta herramienta, aunque útil para el propósito para el que fue diseñada no utiliza ningún método avanzado para representar los individuos. Estos se muestran de forma individual y se utiliza un código de colores para darnos información acerca de cada uno de sus genes. De esta forma es difícil extraer información útil respecto al espacio de búsqueda que es nuestro principal objetivo.

Emma Hart y Peter Ross muestran en (Hart & Ross, 2001) un actualizado estado del arte sobre visualización de algoritmos evolutivos y presentan una nueva herramienta para dicho propósito, **GAVEL**. Esta es la más completa desde el punto de vista del descubrimiento de la cadena de acontecimientos que dan lugar a la obtención de la solución. Su objetivo es visualizar los enlaces evolutivos que se crean durante la evolución de una población. Para su funcionamiento es necesario modificar nuestro algoritmo evolutivo de forma que almacene una serie de datos relativos a la ejecución. Una vez acabada

(a) Vista de una población (VIS) (b) Vista de una ejecución (VIS)

Figura 1.5: Gráficos de la herramienta VIS, de Annie S. Wu. (a) Vista de una población. (b) Vista de una ejecución.

ésta, GAVEL nos permite visualizar las relaciones filogenéticas existentes entre individuos procedentes de diferentes generaciones mediante un árbol de herencia genética. En este árbol, que se centra en el mejor individuo obtenido, podemos ver en qué generación se ha creado cada uno de sus genes y a partir de qué padres y/o operadores genéticos ha sido creado. En la figura 1.6 podemos ver a modo de ejemplo uno de estos árboles creados por el programa. Esta es una forma eficaz de descubrir cómo y por medio de qué operadores se llegan a construir las soluciones óptimas. Como en los casos anteriores no permite obtener información acerca de la forma de un espacio de búsqueda o del área explorada por un algoritmo evolutivo durante su funcionamiento.

Existen otras herramientas que facilitan el proceso de visualización, tales como **GIGA** (Dabs & Schoof, 1995), **Gonzo** (Collins, 1998a), (Wu *et al.* , 1999a) o **GAPlayground** (Dolan, 1998). Todas proporcionan gráficos de evolución del fitness frente al tiempo. La mayoría también muestran información sobre el mejor individuo obtenido en cada generación. Para facilitar información sobre el estado actual del AE, GIGA y GAPlayground permiten volcar la población en forma de matrices de datos sobre ficheros de texto. Estas dos herramientas también muestran información acerca de los fenotipos de los individuos dibujando el camino recorrido por un individuo para

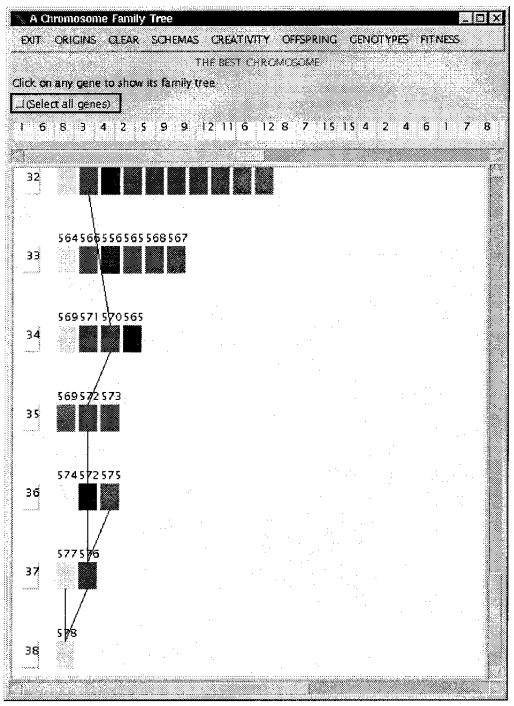


Figura 1.6: Gráficos de GAVEL, de Emma Hart y Peter Ross. Imagen generada por GAVEL que nos muestra la mejor solución encontrada y el árbol de herencia con los ancestros de los que provienen sus componentes.

ciertos tipos de problemas, como, por ejemplo, el TSP. Gonzo es la única de ellas que soporta la visualización del curso de un AE a través del espacio de búsqueda, en el caso de problemas multidimensionales a través de diagramas de dispersión. VIS y GAVEL se centran más en permitirnos seguir la evolución de las estructuras de datos y las relaciones entre individuos que en la representación visual de dichas estructuras. Por desgracia ninguna de ellas permite visualizar el espacio explorado por un algoritmo evolutivo ni el espacios de búsqueda búsqueda en el caso de problemas multidimensionales, cosa que si permite en el caso de problemas de 1 a 3 dimensiones.

1.4. Estructura de la tesis

Tras este capítulo de introducción a la tesis, en el capítulo 2 se describe los algoritmos evolutivos, centrándose en el paradigma de implementación orientado a objetos y se describe la biblioteca Objetos Evolutivos, que será la que utilicemos en los experimentos posteriores.

En el capítulo 3 se hace una clasificación de los métodos de visualización desde el punto de vista de lo que se ve, la estructura de las soluciones o su calidad, y en función del número de generaciones que se visualizan simultáneamente. En su segunda parte se un recorrido por los diferentes métodos de escalado multidimensional que se han ido utilizando desde los años 50 hasta la actualidad.

Tras los anteriores capítulos en que se hace el estudio teórico que soporta esta tesis, en el capítulo 4 se aplica el método de visualización fruto de esta tesis a tres problemas clásicos del campo de la computación evolutiva: el problema onemax, la función de Rastrigín y el problema de la mochila con objeto de descubrir la forma de sus funciones de fitness y averiguar toda la información posible acerca de sus espacios de búsqueda.

En el capítulo 5 estudiamos el efecto de diversos operadores genéticos en visualización. En este caso se ha escogido como a modo de ejemplo un algoritmo evolutivo para entrenar redes neuronales. Para ello se comparan las visualizaciones de varias proyecciones de poblaciones obtenidas alterando las tasas de aplicación de varios operadores genéticos. Al final del capítulo se esboza brevemente lo que son las técnicas de reparto del fitness y se realiza otra serie de experimentos para poder observar su efecto sobre una población de forma visual.

Por último, en el capítulo 6, se muestran las conclusiones y principales aportaciones las que conduce esta tesis junto a una serie de posibles trabajos futuros por donde seguir avanzado en la visualización de algoritmos evolutivos.

1.5. Conclusiones

En este capítulo se han presentado los conceptos básicos en relación a los pilares fundamentales de este trabajo: los algoritmos evolutivos y las técnicas de visualización. También se hace un breve repaso a cómo se han utilizado hasta ahora las técnicas de visualización para ayudar a comprender los algoritmos evolutivos. Por último, se describe la organización seguida para redactar la presente memoria.

Capítulo 2

Algoritmos Evolutivos

Durante la última década los métodos de optimización han cobrado más importancia debido, sobre todo, a que con ellos se pueden resolver ciertos problemas de ingeniería que sólo pueden abordarse mediante aproximación en los computadores actuales.

En el capítulo de introducción se ha descrito someramente la Teoría de la Evolución Natural, y se ha presentado cronológicamente el desarrollo de la informática evolutiva desde sus inicios.

Los paradigmas evolutivos actuales están inspirados en la teoría de la evolución de Darwin (Darwin, 1859) e intentan emular en lo posible a la Naturaleza. De esta forma, los cromosomas y los genes se suelen asociar de alguna forma a cadenas de bits (en AG (Goldberg, 1989; Michalewicz, 1996)) o a vectores de números reales (EE (Schwefel, 1995)). Por otra parte, diversas estrategias de selección (estado estacionario (Whitley, 1989) o (μ, λ) (Schwefel, 1995)) imitan la selección natural.

Sin embargo, la selección natural no es la única forma de selección que podemos encontrar en la Naturaleza. En diversas áreas, como la economía y los mercados, podemos ver cómo ciertas empresas evolucionan, "alimentándose" unas de otras, ocupando *"nichos económicos"*, tal y como describe Brian Arthur en (Arthur, 1990) y John Holland en (Holland, 1975).

Así mismo existe una teoría sobre cómo evolucionan los "universos inflacionarios" (Linde, 1994), según la cual, universos con diferentes leyes físicas

compiten entre sí, resultando ganadores algunos de ellos, ocupando así todo el espacio posible.

Finalmente, incluso las ideas evolucionan (*evolución memética* –memetic evolution– (Heylighen, 1992)), compitiendo por el espacio cerebral, reproduciéndose y mutando: una idea puede ser una canción, una cadena de texto o una imagen de televisión.

En todos estos ejemplos de evolución es difícil identificar un sustrato que evoluciona, pero sí encontramos muchas otras características que nos hacen pensar en ellos como en algún tipo de evolución.

A pesar de que la mayoría de los investigadores afirman que los distintos paradigmas de computación evolutiva sólo difieren en cuanto a la representación y a los operadores de individuo y de población, no existe una visión unificada para todos ellos, y mucho menos una herramienta que lo unifique.

En este capítulo se introducirán los algoritmos evolutivos, centrándonos en la abstracción denominada ***"Objetos Evolutivos"*** (en lo sucesivo OE), que engloba todos los paradigmas de la computación evolutiva, abstrayendo las características comunes a todos los paradigmas.

En la siguiente sección (2.1) se lleva a cabo una revisión de los principales métodos utilizados para resolver problemas de optimización, estudiando los procedimientos de escalada (sección 2.1.1) y el enfriamiento simulado (sección 2.1.2), antes de introducir los algoritmos evolutivos (sección 2.1.3).

En la sección 2.2 se introduce el concepto de Computación Orientada a Objetos, para pasar a presentar la abstracción de Objetos Evolutivos describiendo sus fundamentos y sus principales características (secciones 2.2.1 y 2.2.2).

La sección 2.3 estudia a fondo los Objetos Evolutivos, describiendo cómo diseñar un AE basado en OE, enumerando cada elemento que lo compone y describiéndolos posteriormente.

Para terminar, en la sección 2.4 se aplicarán los métodos definidos en el capítulo a la resolución de varios ejemplos "tipo" usando algoritmos evolutivos basados en OE.

2.1. Los Algoritmos Evolutivos en el Contexto de los Problemas de Búsqueda y Optimización

Antes de entrar con detalle en la descripción de la abstracción de Objetos Evolutivos conviene situar los algoritmos evolutivos en el marco de los problemas de búsqueda y optimización, comparándolos con otros métodos de optimización bien conocidos, como los procedimientos de escalada y de enfriamiento simulado.

2.1.1. Procedimientos de escalada

Los *métodos indirectos de resolución* buscan el óptimo llevando iterativamente el valor de la función objetivo en la dirección de máxima pendiente. Procediendo de este modo se garantiza encontrar un óptimo local o *subóptimo* (un máximo o mínimo, esto es, un punto rodeado de puntos peores que él), aunque no necesariamente el óptimo global.

Este **procedimiento de escalada** ("hillclimbing") da lugar a algoritmos de resolución muy sencillos, siempre y cuando se disponga de una buena técnica de determinación del signo de la pendiente. Estos métodos usan una técnica iterativa de mejora, que es aplicada a un sólo punto (el punto actual) en el espacio de búsqueda. En cada iteración se determina una región de máxima 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. El método termina cuando no sea posible ninguna mejora.

En la figura 2.1 se muestra un algoritmo simple de optimización por escalada.

Esta técnica tiene dos serias desventajas, que van en detrimento de su robustez. Primero, admite, de un modo u otro, que aunque no se pueda

```
seleccionar un punto de partida al azar X_0
for t=0 to LimIteraciones
        E_t = EvaluarFuncionObjetivo(X_t)
        parar = FALSE
        while (NOT parar)
                P_t = seleccionar un vecino de X_t
                N_t = Evaluar(P_t)
                if (N_t > E_t) then
                        X_t = P_t
                        E_t = N_t
                else
                        parar = TRUE
        endwhile
endfor
```

Figura 2.1: Estructura general de un algoritmo de escalada.

dar una forma cerrada a la función objetivo, tiene sentido el concepto de pendiente; ello restringe su campo de aplicación; además, requiere 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, es esencialmente local, esto es, sólo puede hallar subóptimos y, lo que es peor, en el caso de que haya varios (existe *multimodalidad*) el acabar en uno u otro depende del punto inicial, siendo imposible determinar a priori a qué subóptimo llevará un punto inicial dado. Es más, no hay información acerca del error relativo cometido con respecto al óptimo global. Esas dos características se resumen en que, por un lado, esta técnica de optimización es muy eficiente pero por otro muy específica. Las desventajas de la alta especificidad no se logran compensar con la alta eficiencia, lo que la hace poco robusta.

2.1.2. Enfriamiento simulado

La técnica de **enfriamiento simulado** ("simulated annealing") (Korst, 1989) es un método inspirado en la Naturaleza. Básicamente es una analogía del modo en que los materiales solidifican en un cristal según se van enfriando, es decir, reduciendo su energía hasta un mínimo de energía, ofreciendo máxima resistencia, y la búsqueda del óptimo de un problema. Metropolis et al. (Teller, 1958) propusieron el método para encontrar la configuración en equilibrio de una colección de átomos a una temperatura dada. La conexión entre este algoritmo y la minimización matemática fue propuesta por Kirkpatrick et al. (Vecchi, 1983) como una técnica de optimización para problemas combinacionales y otros tipos de problemas. Esta técnica evita muchas de las desventajas de los métodos de escalada: la solución no depende del punto de inicio. Esto se consigue introduciendo una probabilidad de aceptación del nuevo punto, que será 1 si el nuevo punto mejora al antiguo, o bien dependerá de la diferencia entre los valores de bondad de ambos puntos, y de un nuevo parámetro llamado, por su analogía al símil físico, temperatura. A menor temperatura, menor es la probabilidad de aceptar un nuevo punto. Durante la ejecución del algoritmo, la temperatura decrece, y termina cuando se alcanza un valor de la temperatura para el cual, virtualmente ya no se van a aceptar más cambios.

En un problema típico de enfriamiento simulado, se define la función a minimizar (*función de coste*, f), y después, a partir de una solución aleatoria inicial, se van generando k diferentes soluciones (llamado número de cambios) que se comparan con la solución que en ese momento es la actual. La mejor solución encontrada (menor coste) se adopta como siguiente actual, aunque una solución de mayor coste (peor) se adoptará con una probabilidad que irá decreciendo con el tiempo, de acuerdo al valor de la temperatura.

La búsqueda realizada por el algoritmo actúa del siguiente modo: se genera un punto o estado, s_n, a partir del anterior, s_0, utilizando un operador de cambio (generador de nuevos estados). Si el nuevo estado es mejor, se acepta.

Si el nuevo es peor, se aceptará con una probabilidad de:

$$p_a = e^{-\frac{\Delta f(s)}{T}} \tag{2.1}$$

donde $\Delta f(s)$ se define como el incremento de la función de coste y T es la temperatura. Dicho parámetro se va decrementando paulatinamente mediante una *función de reducción de temperatura*. La más simple es:

$$T_{n+1} = \alpha T_n \tag{2.2}$$

donde α es una constante menor que 1. Este *esquema exponencial de enfriamiento* fue propuesto por Kirkpatrick et al. (Vecchi, 1983) usando $\alpha = 0{,}95$

Kirkpatrick (Kirkpatrick, 1984) sugirió que T_0 depende de f de acuerdo a:

$$T_0 = -\frac{\Delta f^*}{lnp_a} \tag{2.3}$$

donde Δf^* es el incremento en la función objetivo observado y p_a es la probabilidad inicial de aceptación de la solución (se suele utilizar 0,8).

En la figura 2.2 se muestra una versión del algoritmo de enfriamiento simulado descrito en (Michalewicz, 1996).

2.1.3. Algoritmos evolutivos

Los **algoritmos evolutivos** son una alternativa para resolver problemas de optimización complejos. Aunque pertenecen a la clase de algoritmos probabilísticos, son muy diferentes de los algoritmos aleatorios, ya que combinan elementos de búsqueda estocástica y directa. Es por esto que son mucho más robustos que los algoritmos existentes de búsqueda directa. Otro punto a favor de los AE es que mantienen una población de posibles soluciones, mientras que otros métodos se centran y procesan un sólo punto del espacio de búsqueda, descartando los demás.

Los AE llevan a cabo una búsqueda multidireccional manteniendo una

```
n = 0
inicializar la temperatura T
seleccionar aleatoriamente un estado s_o
calcular f(s_o)
repeat
        for j=1 to k
                seleccionar un nuevo estado s_n en la vecindad
                        de s_o modificando s_o
                calcular f(s_n)
                Δf = f(s_o) − f(s_n)
                if (Δf < 0) OR (random(0,1) < e^(−Δf/T))
                        then s_o = s_n
        endfor
        T = f_T(T, n)
        n = n + 1
until (T < T_min)
```

Figura 2.2: Estructura general de un algoritmo de Enfriamiento Simulado.

población de soluciones potenciales, creando información y haciendo un intercambio de ésta en todas las direcciones del espacio de búsqueda. La población simula una evolución: en cada generación las mejores soluciones se reproducen, mientras que las peores, desaparecen. Para distinguir entre buenas y malas soluciones, utilizamos una función objetivo (de evaluación o "fitness") que mide la calidad de la solución (la aproximación de un individuo a la solución óptima).

2.2. Computación Evolutiva Orientada a Objetos

Uno de los libros más conocidos sobre Computación Evolutiva es *"Genetic Algorithms + Data Structures = Evolution Programs"* (Michalewicz, 1996). La propuesta que hace Michalewicz sirve como idea inicial para introducir el

concepto de Objeto Evolutivo (OE), en cuanto a que OE pretende eliminar el problema de elegir una representación para resolver un problema y a la vez posibilitar la aplicación del paradigma principal de la programación procedural a la computación evolutiva: los *algoritmos* se aplican a las *estructuras de datos*, ambos son distintos y deben estar separados, y juntos forman los programas evolutivos (Merelo *et al.* , 1999; Schoenauer, 2000; Merelo *et al.* , 2000).

Sin embargo, los lenguajes de más uso en la actualidad se basan en la programación orientada a objetos. Así, en computación orientada a objetos, los algoritmos y las estructuras de datos a las que se aplican aquellos, se encuentran encapsuladas en objetos, y la interacción entre dichos objetos se regula mediante una interfaz. Un objeto conoce su funcionamiento interno, pero quien quiera utilizarlo (el *cliente* de ese objeto) deberá hacerlo a través de la funcionalidad asociada a la interfaz externa del objeto.

A continuación se presentará el concepto de Objeto Evolutivo, describiendo las características fundamentales que lo diferencian de otros paradigmas de computación evolutiva.

2.2.1. Objetos Evolutivos

Como se comentaba anteriormente, OE define interfaces para todos los objetos, de forma que quede restringido el modo en que unos usan a los otros.

A pesar de las restricciones que introducen dichas interfaces, éstas deben dotar a los objetos de ciertas características básicas para poder hacerlos evolucionar. Estas características fundamentales incluyen que un objeto sea **replicable, mutable, combinable** y **comparable**. Estas propiedades se usarán como analogía computacional para los tres criterios evolutivos descritos por Maynard (Maynard-Smith, 1997) (heredabilidad, variabilidad y fecundidad). Examinemos más detalladamente cada una de ellas:

- **Replicabilidad**. Es necesario poder obtener copias de un OE cualquiera, ya sea por él mismo o mediante el uso de otros objetos (*replicadores*). También debería ser posible crear OE desde cero (a través de "fac-

torías" de objetos).

- **Mutabilidad**. Un OE debe poder mutar o modificarse de forma que se incremente la diversidad de un conjunto (población) de OE. Esto quiere decir que el OE, por sí mismo o mediante un objeto cuyo cometido será modificar otros objetos (*objeto mutador*), podrá cambiar de diversas formas, pero el funcionamiento de dichos cambios no tiene por qué conocerse desde fuera, ni tampoco tiene por qué haber una representación específica para permitir una mutación de un objeto. Básicamente el "cliente" sólo necesita saber que el objeto mutado cambiará de alguna forma.

- **Combinabilidad**. En algunos casos convendrá combinar dos o más OE para crear uno nuevo (de forma similar a como funciona el cruce en los AG). Aunque esto no será posible en todas las ocasiones, cuando lo sea, la operación hará decrecer la diversidad, en el sentido que hace los OE más parecidos entre sí. Tal y como ocurre con la mutación, el funcionamiento interno del cruce no tiene por qué ser conocido por el "cliente".

- **Comparabilidad**. Para decidir qué objetos son mejores que otros al realizar una tarea en particular, deberían poderse comparar para decidir cuál de ellos es el mejor. Esto no significa que cada OE deba tener un valor escalar de la función de evaluación (para llevar a cabo una selección proporcional al valor de la función de evaluación), ni siquiera que deba tener un valor explícito, simplemente debe poder determinarse cual es el mejor entre dos. Así, un objeto *selector* puede eliminar el peor y reproducir el mejor (mutándolo o cruzándolo con otros).

2.2.2. Innovaciones introducidas por Objetos Evolutivos

Las principales diferencias que encontramos entre OE y otros paradigmas de computación evolutiva son (Merelo *et al.* , 1999; Schoenauer, 2000; Merelo

et al. , 2000):

- OE es *independiente del paradigma*, lo cual quiere decir que no es un algoritmo genético, ni un programa evolutivo, ni un programa genético, ni un enfriamiento simulado, pero que puede ser cualquiera de ellos, y de hecho todos esos paradigmas de la computación evolutiva pueden implementarse con OE.

  Por ejemplo, para implementar un AG simple, el objeto mutador simplemente debería cambiar el valor de ciertos bits en la cadena; el objeto reproductor aplicaría el procedimiento del operador de cruce; una función de evaluación escalar permitiría la comparación de dos OE, y a partir de éste se podría aplicar un proceso de selección (ruleta) para reproducir los mejores y eliminar los peores individuos.

- OE es *independiente de la representación*. OE no necesita información acerca de cómo se representa una solución a un problema, sino simplemente del hecho de que éstas pueden cambiarse y si es conveniente, combinarse entre sí. De hecho, OE no necesita representar las soluciones de un modo diferente, simplemente codificarlas en el lenguaje orientado a objetos: una solución a un problema es un objeto (perteneciente a cierta clase), que podrá ser mutado, combinado con otros de su clase y comparado para estimar cuál es el mejor, lo cual es suficiente para hacerlo evolucionar.

- OE es *independiente del lenguaje de programación*, lo cual quiere decir que no necesita codificar las soluciones como objetos de un lenguaje orientado a objetos en particular, tal y como el paradigma de la Programación Genética hace (representa las soluciones en LISP). Las soluciones se pueden programar en cualquier lenguaje orientado a objetos y hacerlos evolucionar utilizando la biblioteca de OE del mismo lenguaje. De hecho, los OE se pueden programar en diferentes lenguajes de programación y hacerlos evolucionar a la vez, usando modelos de objetos independientes del lenguaje de programación, tales como *COM*

-*Common Object Model*- de Microsoft (Microsoft, 2000) o *CORBA* -*Common Object Request Broker Architecture*- del Object Management Group (Henning & Vinoski, 1999; OMG, 2000).

- OE es *orientado a objetos desde la base*: no sólo el objeto o la clase del individuo que se hará evolucionar se programa según este paradigma de programación, sino también cualquier elemento que intervenga en la evolución (operadores, selectores y los algoritmos mismos).

OE hace énfasis en la modularidad a la hora de programar una aplicación evolutiva. Debido a que tanto los operadores como el resto de entidades son objetos, resulta fácil combinar diferentes operadores y objetos de distintas fuentes para construir una aplicación. En la práctica no importa el interior del objeto, sino cómo se pueda acceder a él mediante su interfaz.

2.3. Diseño y Desarrollo de Aplicaciones con Objetos Evolutivos

Objetos Evolutivos es una herramienta, desarrollada en C++ que define las interfaces para varias clases de algoritmos usados en computación evolutiva y al mismo tiempo, implementa dichos algoritmos. La biblioteca se encuentra disponible en Internet en http://sourceforge.net/projects/eodev.

El desarrollo de cualquier programa conlleva dos fases: *el diseño* y *la programación*. Utilizando OE para desarrollar una aplicación evolutiva nos encontramos con estas dos fases, en las cuales primero se decide una representación adecuada al problema y los operadores genéticos a utilizar, y después se diseña un algoritmo evolutivo, utilizando los elementos anteriores del modo en que se haría cualquier otro algoritmo evolutivo.

Durante la **fase de diseño**, normalmente el *objeto que se hará evolucionar* queda definido por el problema a resolver. Por ejemplo, en el caso del problema del viajante de comercio ("Travelling Salesman Problem", TSP) se podría utilizar una representación de modo que cada objeto solución sea

un vector de enteros que contenga todas las ciudades (Booch, 1994). Los *operadores genéticos* utilizados para mutar y combinar las soluciones deberían tener en cuenta la disposición de las ciudades a visitar. El resto del diseño es, en general, independiente del problema: la *selección* de los individuos que se *reproducirán*, y el *criterio de parada del algoritmo*.

En la **fase de programación**, el algoritmo (basado en OE) se programará de forma similar a como se programaría otro algoritmo evolutivo, esto es, según un bucle del tipo:

1. Crear una población inicial de OE (usando factorías o los constructores de cada objeto).

2. Repetir hasta que el criterio de parada se cumpla:

   a) Evaluar cada individuo, comparándolos entre sí, para seleccionar los mejores.

   b) Aplicar los operadores genéticos, creando nuevas soluciones (incrementar la diversidad) y combinándolas (decrementar la diversidad).

   c) Sustituir los peores individuos en la población por los recién creados.

Aunque no hay un soporte teórico para probar el buen funcionamiento de OE, en el peor de los casos funcionaría como una búsqueda aleatoria, pero ya que utiliza recombinación de soluciones, los "bloques constructivos" que producen buenas soluciones se mezclan, con lo cual se alcanzan mejores soluciones.

Mientras que la codificación binaria está respaldada por el *"Teorema de los Esquemas"* (Holland, 1975; Goldberg, 1989), la *"Teoría del Análisis de Formas"* (Radcliffe, 1991) da soporte teórico al uso de alfabetos no binarios. El Análisis de Formas generaliza el Teorema de los Esquemas tratando con la equivalencia de clases de cromosomas en lugar de cromosomas binarios. Un ejemplo es "cromosomas que contienen una permutación en particular" o "cromosomas en los que el segundo componente es el doble que el primero".

El Análisis de Formas asegura que si se usan operadores independientes de la representación se pueden definir algoritmos independientes de la representación, que se comportarán de la misma forma que los AG estándar. En OE, puesto que el usuario no está obligado a utilizar una representación en cadenas binarias, y las soluciones a los problemas se codifican de forma *natural*, los bloques que forman las soluciones se encontrarán juntos de forma natural y se heredarán juntos.

2.3.1. Individuos de la población

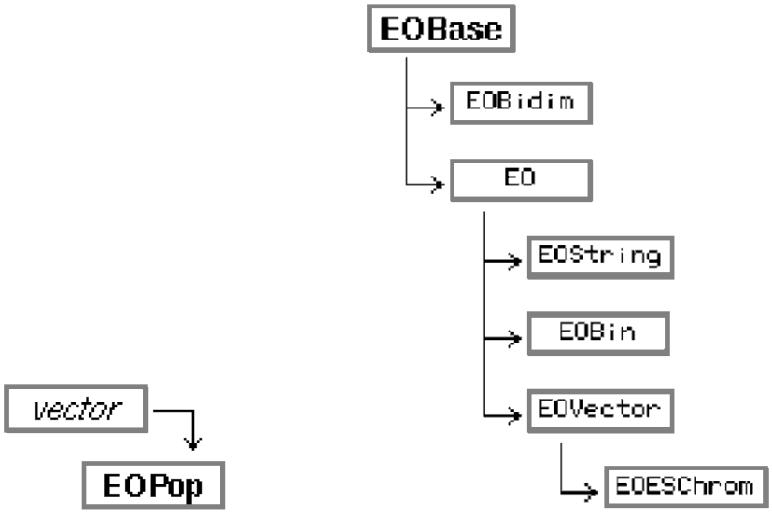
La biblioteca proporciona ciertas clases de objetos que actúan como individuos en la población, es decir, objetos evolutivos. Como se ha venido comentando, la biblioteca incluye diversos OE, entre los que se encuentran: cadenas de bits, cadenas de caracteres, vectores de cualquier tipo de dato (en particular, de números enteros o reales), y por supuesto, MLP.

La figura 2.3 representa la jerarquía de clases de la versión de la biblioteca OE utilizada en el desarrollo de esta tesis (versión 0.8.5) para las poblaciones y los cromosomas, que incluye una clase genérica de cromosomas (**EOBase**, que define la interfaz básica para un OE, incluyendo una función de evaluación y un identificador) y algunos tipos más específicos (**EOBin, EOString, EOVector**). Se pueden obtener OE nuevos, con las características fundamentales ya comentadas, simplemente heredando alguna de las clases ya definidas (Merelo *et al.* , 1999; Schoenauer, 2000; Merelo *et al.* , 2000).

2.3.2. Operadores genéticos

Los operadores genéticos se encuentran agrupados en dos clases: genéricos y específicos.

Los **operadores genéricos** se pueden aplicar a muchas clases de OE, sin importar si estos hacen uso de algún tipo de representación. Por ejemplo, un *operador de permutación* se puede aplicar a cualquier objeto de una clase que herede de un vector genérico (**EOVector**), del mismo modo que se puede



(a) Jerarquía de poblaciones (b) Jerarquía de cromosomas

Figura 2.3: Jerarquía de clases de la biblioteca OE para las poblaciones (a) y para los cromosomas (b).

aplicar un *operador de cruce*.

Otros operadores, como el de **mutación** son más específicos del OE a modificar: un objeto mutador de cadenas de bits operará de muy diferente modo a como lo hará un objeto mutador de vectores de números enteros.

Los operadores son objetos cuyo comportamiento es el de una función a la que se pasa el objeto a modificar.

La figura 2.4 representa la jerarquía de clases de la biblioteca OE para los operadores genéticos, que incluye operadores *unarios* (como el de mutación), *binarios* (como el de cruce), o *n-arios* (aplicados a más de dos OE (Eiben *et al.* , 1995)). Esas clases definen las características básicas de cada tipo de operador. De ellas descienden los operadores que se utilizarán en las aplicaciones.

El tener los operadores diferenciados de los OE permite trabajar más fácilmente con ellos, ya que al estar definidos como clases independientes de las que se puede heredar, el usuario puede crear sus propios operadores.

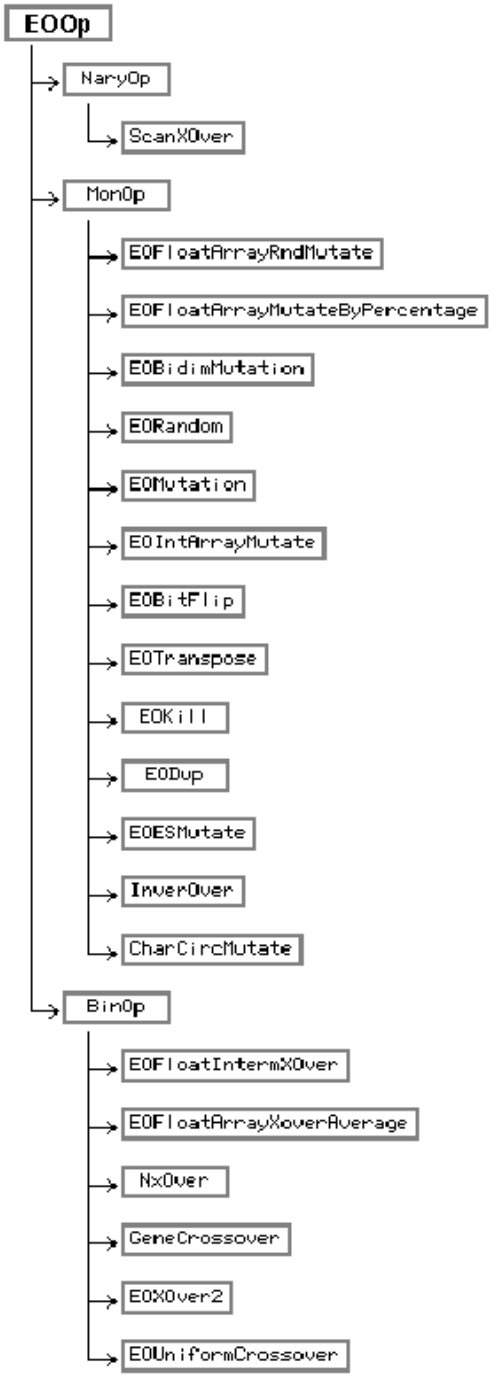


Figura 2.4: Jerarquía de clases de la biblioteca OE para los operadores genéticos.

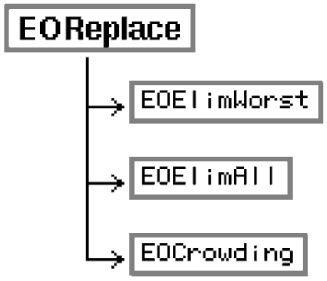


Figura 2.5: Jerarquía de clases de la biblioteca OE para los operadores de reemplazo.

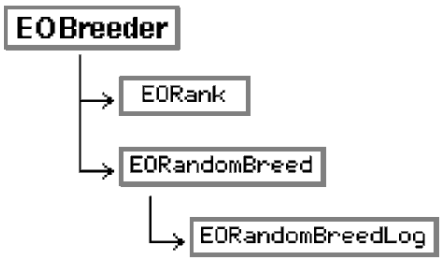


Figura 2.6: Jerarquía de clases de la biblioteca OE para los operadores de crianza.

2.3.3. Operadores de población

Los operadores de población son genéricos y actúan con varios objetos de la población (seleccionando algunos de ellos, creando individuos a partir de otros o sustituyendo algunos individuos por otros).

Entre estos operadores se incluyen el de selección, crianza y reemplazo.

Las figuras 2.5, 2.6 y 2.7 representan la jerarquía de clases de la biblioteca OE para los operadores de población, que incluye operadores de *selección*, *crianza*, o *reemplazo*.

Los objetos **EOReplace** (y descendientes) reemplazan, eliminando si es necesario, aquellos individuos de la población peor dotados por los creados con los operadores genéticos.

Un objeto de la clase **EOBreeder** (y descendientes) almacena los oper-

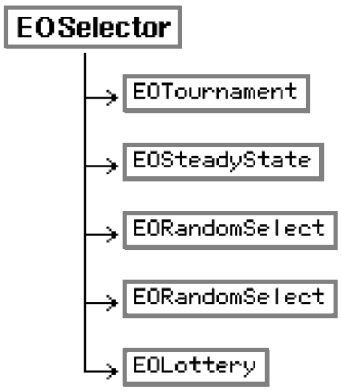


Figura 2.7: Jerarquía de clases de la biblioteca OE para los operadores de selección.

adores genéticos para aplicarlos posteriormente a los individuos seleccionados, creando la descendencia de esa generación. Cada operador recibe como parámetro una *prioridad de aplicación*, y en cada generación, antes de aplicar los operadores genéticos, las prioridades de todos los operadores se normalizan dividiéndolas por la prioridad total acumulada para obtener la *probabilidad de aplicación* de cada operador.

Un **EOSelector** (y descendientes) lleva a cabo la selección de los individuos que se reproducirán aplicándoles los operadores genéticos, de acuerdo a cierta estrategia (torneo, estado estacionario).

2.3.4. Algoritmos

En OE, los algoritmos son tratados también como objetos, de forma que en cualquier momento podemos crear un nuevo objeto de tipo algoritmo, acceder a través de su interfaz a sus componentes internos (objetos población, selector, reproductor, terminador).

Los algoritmos se aplican a una población de OE, haciéndolos evolucionar hasta que se alcanza una condición. Básicamente llevan a cabo el bucle de evaluación, selección, reproducción y sustitución.

La implementación actual de la biblioteca incluye los siguientes objetos

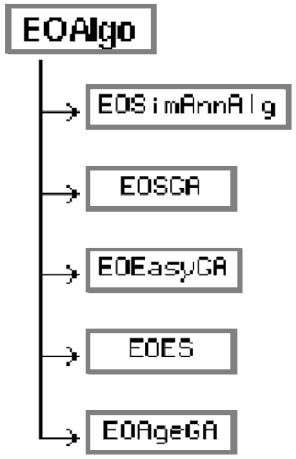


Figura 2.8: Jerarquía de clases de la biblioteca OE para los algoritmos.

algoritmo: **EOES** (estrategias de evolución), **EOEasyGA** (un AE simple y flexible), **EOSGA** (el AG de Goldberg (Goldberg, 1989)), y **EOSA** (enfriamiento simulado).

La figura 2.8 representa la jerarquía de clases de la biblioteca OE para los algoritmos comentados.

2.4. Comportamiento de un AE basado en OE ante Problemas Tipo

En esta sección se estudiarán varios ejemplos de optimización de diversas funciones mediante un AE basado en OE. Se trata de aclarar y concretar todo lo comentado a lo largo del capítulo con varios ejemplos prácticos del funcionamiento de un AE para optimización de funciones reales multimodales.

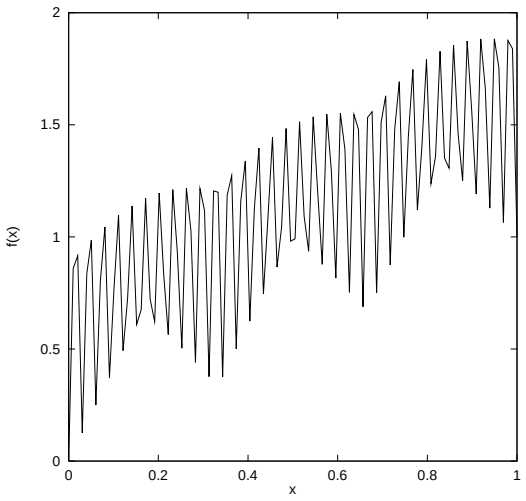


Figura 2.9: Gráfica de la función objetivo $f(x) = x + |sin(32\pi x)|$

2.4.1. Función real multimodal de una variable

El primer ejemplo de esta sección tratará de encontrar el óptimo (máximo en este caso) de la siguiente función, propuesta por Riolo (Riolo, 1987):

$$f(x) = x + |sin(32\pi x)| \tag{2.4}$$

cuya gráfica es la que se muestra en la figura 2.9. El problema está en encontrar la x en el intervalo $[0 \dots 1]$ que maximice la función f, esto es, encontrar un x_0 tal que

$$f(x_0) > f(x) \qquad \forall x \in [0 \dots 1]$$

Si observamos la gráfica de la función (figura 2.9) vemos que la función es fuertemente multimodal, esto es, en el intervalo $[0 \dots 1]$ tiene 32 óptimos locales, por lo cual, un método analítico de optimización tenderá a estancarse en alguno de ellos, no llegando al óptimo global ($x = 0{,}98447395, f(x) = 1{,}98442447$) si no se le da una aproximación inicial lo suficientemente buena.

Para resolver este problema se ha desarrollado un programa, utilizando la biblioteca OE, lo cual simplemente requiere definir la clase de los individuos, programar la función de evaluación, y elegir los objetos transformadores

```
struct funcEvaluacion: public EOEvalFunc< Chrom > {
  virtual float evaluate( Chrom & eo ) const {
    EOView<float> v(eo, 0,1);
    float f;
    float x = v.readGene(0);
    f = x + fabs( sin( 32 * 3.1415927 * x ) );
    eo.fitness( f );
    return f;
  };
};
```

Figura 2.10: Codificación de la función de evaluación para resolver el problema de la función Riolo.

(operadores genéticos), selectores, terminadores, y el tipo de algoritmo que se utilizará.

Como algoritmo principal del programa se eligió un algoritmo genético simple (clase **EOEasyGA**), que básicamente lo que hace es el bucle de un AE, aplicando selección, reproducción/mutación y reemplazo.

La clase a la que pertenecen los individuos es **EOBin**, lo cual significa que los individuos serán cadenas de bits (como en un AG).

El objeto selector es un algoritmo de estado estacionario (clase **EOSteadyState**).

Como operadores de transformación se ha hecho uso de los operadores de mutación (**EOMutation**) y de cruce (**EOXOver2**).

La condición de parada del programa se estableció mediante el uso de un terminador (clase **EOGenTerm**) que controla el número de iteraciones que se harán.

Por último, la función de evaluación viene dada por la ecuación 2.4, de forma que, implementando dicha función, como una clase descendiente de **EOEvalFunc** (la que define la interfaz de cualquier función evaluadora), como se muestra en la figura 2.10, ya tendremos el programa listo para utilizarlo.

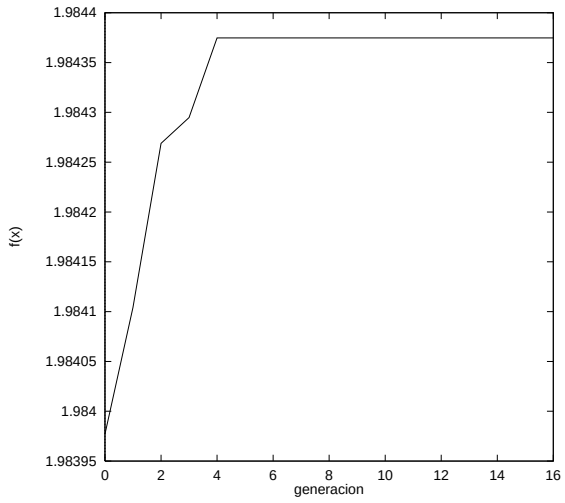


Figura 2.11: Ejecución típica del AE para optimizar la función Riolo: En las primeras generaciones los individuos quedan bastante alejados de la solución. Conforme avanza la ejecución (generaciones 2 y 3), el AE encuentra individuos que mejoran la solución. Hacia la generación 4, se encuentra un individuo que representa una solución adecuada.

En la figura 2.11 podemos observar cómo se desarrolla la ejecución del AE implementado que optimiza esta función. Vemos que en las primeras generaciones los individuos quedan bastante alejados de la solución. Conforme avanza la ejecución (generaciones 2 y 3), el AE encuentra individuos que mejoran la solución. Hacia la generación 4, se encuentra un individuo que representa una solución lo suficientemente buena como para, pasadas unas generaciones sin obtener una mejora, conformarnos con la solución obtenida y detener la ejecución.

2.4.2. Función real de dos variables

En este otro ejemplo se trata de encontrar el óptimo (máximo en este caso) de la siguiente función:

$$f(x,y) = \frac{sin(\sqrt{x^2+y^2})}{\sqrt{x^2+y^2}} \tag{2.5}$$

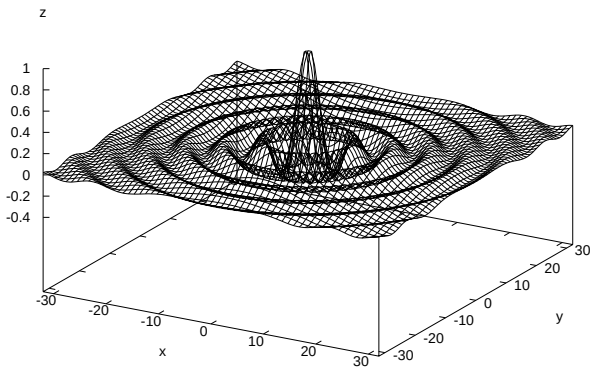


Figura 2.12: Gráfica de la función "marea".

cuya gráfica es la que se muestra en la figura 2.12. El problema está en encontrar las coordenadas (x, y) que maximicen la función $f(x, y)$.

Si observamos la gráfica de la función (figura 2.12) vemos que es multimodal: la forma de la función va haciendo ondulaciones conforme se acerca a la coordenada $(0, 0)$, alcanzando el óptimo global en la coordenada $(0, 0)$ de forma que $f(x, y) = 1$.

Para resolver este otro problema se ha modificado el programa desarrollado en la sección anterior. Simplemente ha habido que cambiar la función de evaluación y la longitud de los individuos (para representar dos números reales en lugar de uno).

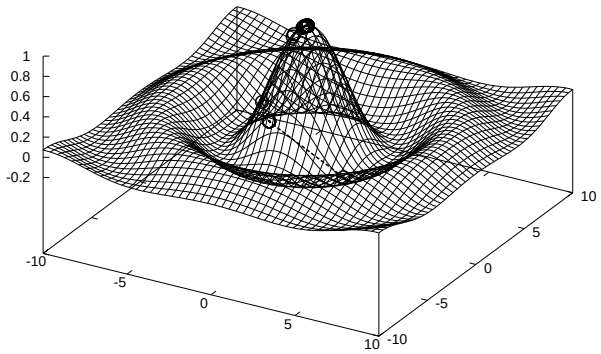
La función de evaluación viene dada por la ecuación 2.5, así simplemente tenemos que implementar dicha función, como una clase descendiente de **EOEvalFunc**, como se muestra en la figura 2.13.

En la figura 2.14 podemos observar cómo se desarrolla la ejecución del AE que optimiza esta función. En (a) podemos ver la trayectoria de la búsqueda realizada por la superficie tridimensional; (b) representa la proyección en el plano XY del camino seguido en dicha búsqueda. Al principio el AE realiza la

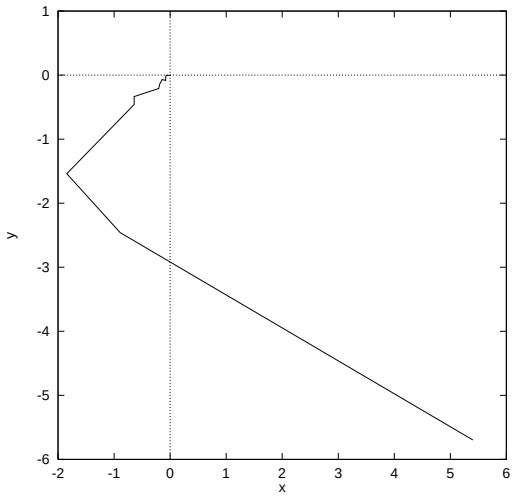
```
struct funcEvaluacion: public EOEvalFunc< Chrom > {
  virtual float evaluate( Chrom & eo ) const {
    EOView<float> v(eo, 0,1);
    float f;
    float x = v.readGene(0), y=v.readGene(1);
    if((x!=0)&&(y!=0)){
      result = sqrt( x*x + y*y ) ;
      f = sin(result) / result ;
    }else{
      f=1;
    }
    eo.fitness( f );
    return f;
  };
};
```

Figura 2.13: Codificación de la función de evaluación para resolver el problema de la función marea.

búsqueda lejos del óptimo (las soluciones que encuentra están por el $(6, -6)$). Conforme avanza en la ejecución, la búsqueda se va acercando poco a poco al óptimo: busca hacia el $(0, -3)$, después por el $(-2, -1{,}5)$, para acabar dirigiendo la búsqueda hacia el $(0, 0)$.



(a) superficie tridimensional de la función marea



(b) proyección en el plano XY del camino seguido en la búsqueda

Figura 2.14: Ejecución típica del AE para optimizar la función "marea". (a) representa la búsqueda realizada por la superficie tridimensional; (b) representa la proyección en el plano XY del camino seguido en dicha búsqueda.

2.5. Conclusiones

En este capítulo se han presentado los AE como una potente herramienta para resolver problemas de ingeniería. Son una rama de la computación que explora nuevas aproximaciones inspiradas en la biología. Los sistemas computacionales que se necesitan actualmente para resolver ciertos problemas requieren adaptabilidad, paralelismo, capacidad de aprendizaje y creatividad, lo cual nos hace pensar que debemos imitar los sistemas naturales que posean dichas características.

Se ha aplicado la idea de "evolución" más allá de la Naturaleza y de los paradigmas computacionales evolutivos clásicos, apuntando las limitaciones que presentan hoy día la mayoría de las aproximaciones que intentan ofrecer una visión unificada de todos esos paradigmas.

Se ha presentado una nueva abstracción para computación evolutiva, que denominaremos "Objetos Evolutivos" (OE), basada en la programación orientada a objetos y que reune las características comunes a todos los paradigmas evolutivos.

OE hace énfasis en la modularidad a la hora de programar una aplicación evolutiva. Debido a que tanto los operadores como el resto de entidades son objetos, resulta fácil combinar diferentes operadores y objetos de distintas fuentes para construir una aplicación determinada.

OE prescinde de la necesidad de representar las soluciones, haciendo evolución de objetos complejos directamente usando operadores genéticos, y estableciendo, mediante interfaces, el modo en que unos objetos acceden a otros.

Se ha mostrado la facilidad de uso de OE y de la robustez y capacidad de resolución que presenta, resolviendo dos problemas clásicos de búsqueda en superficies multimodales con sendos programas desarrollados con la biblioteca presentada.

Capítulo 3

Técnicas de Visualización

Los **Algoritmos Evolutivos** (AE) suelen producir una gran cantidad de datos durante su ejecución. Comprender su comportamiento suele ser difícil debido a su forma iterativa y estocástica [1] de buscar soluciones. Además de la convergencia hacia la solución, la extracción de información interesante acerca de su estado y su curso no es una tarea trivial, de forma que la visualización puede ser una herramienta muy útil para la comprensión y el uso efectivo de dichos algoritmos. La aplicación de técnicas de visualización al mundo de la AE ha recibido una atención creciente en los últimos años; durante mucho tiempo ha sido utilizada sólo de una manera parcial para seguir la evolución de la calidad de las soluciones, pero no la de los componentes de dichas soluciones, y otros aspectos como la exploración del espacio de búsqueda.

Los usuarios de AE suelen observar cómo mejora la calidad de las soluciones a lo largo del tiempo mediante gráficos que muestran el cambio del fitness a lo largo de las generaciones. Aunque tales gráficos muestran la mejora en la calidad de las soluciones frente al tiempo, no dice nada respecto a la estructura de dichas soluciones ni acerca de las regiones del espacio de búsqueda que están siendo exploradas. Las técnicas de visualización son una potente herramienta para resolver este problema. Su uso puede facilitar la comprensión de su funcionamiento de forma que los usuarios de AE podrán

[1] dependiente de variables aleatorias

confiar en los componentes de sus algoritmos y en los valores de sus parámetros.

En el resto del capítulo se tratarán los siguientes temas: En la sección 3.1 se presentan varias formas de clasificar las técnicas de visualización en función de que tipo información se visualicen o de la cantidad de información visualizada. En la sección 3.1.1 se explicarán las diferencias entre genotipo y fenotipo, así como las ventajas que pueden aportarnos la visualización de cada una de ellas. En la sección 3.1.2 se discuten las ventajas e inconvenientes de primar la cantidad de información a visualizar frente a la calidad de la misma y viceversa. Por último en la sección 3.2 haremos un recorrido por las distintas técnicas de escalado multidimensional existentes y analizaremos sus ventajas e inconvenientes.

3.1. Clasificación de técnicas de visualización

Las técnicas de visualización han sido empleadas siempre que ha sido posible debido a su capacidad descriptiva. Como dice un viejo refrán, más vale una imagen que mil palabras. Estas técnicas son una herramienta esencial para descubrir el funcionamiento del proceso evolutivo. Emma Hart y Peter Ross en (Hart & Ross, 2001) muestran el estado del arte sobre visualización de algoritmos evolutivos. Hartmut Pohlheim en (Pohlheim, 1999) hace un repaso por algunas de las técnicas más difundidas y las implementadas en forma de *toolbox* para Matlab (The MathWorks, Inc, 1994). Los diversos métodos de visualización de algoritmos evolutivos pueden ser divididos en diferentes categorías de acuerdo a varios criterios:

- *¿Qué se ve?* Según este criterio de clasificación puede observarse el **genotipo**, la representación del problema, o el **fenotipo**, la calidad de la solución que se induce de dicha representación. Cada uno de ellos permite acercarnos a un tipo de información diferente, con lo cual, ninguno de los dos métodos puede considerarse mejor ni peor sino en función de nuestro interés en cada caso en particular. Así, si estamos interesados en conocer cómo son las soluciones, la visualización del genotipo

resolverá nuestro problema. Si por el contrario lo que nos interesa es la calidad de las soluciones, la visualización del fenotipo será más adecuada. Casi siempre es deseable poder aunarlas para obtener más y mejor información acerca de nuestro algoritmo, pero normalmente no es posible y habrá que establecer un compromiso en cuanto a qué es más importante. En la figura 3.1 puede verse un ejemplo de la ejecución de un AE para la resolución del problema del viajante de comercio. En dicha figura, se muestra la evolución tanto de los genotipos como de los fenotipos de toda un población. Las 5 primeras imágenes muestran representaciones bidimensionales de la población de individuos en diversas etapas de la evolución, de sus genotipos, desde el estado inicial hasta encontrar la solución óptima alrededor de la generación número 70. En el último gráfico se puede ver la evolución de la calidad de las soluciones, es decir, de los fenotipos frente al tiempo.

- *¿Cuánto se ve?* Podemos ver datos procedentes de una generación o de varias generaciones a la vez. Cuando observamos datos generados en una sola generación podemos hacernos una imagen del estado del AE en un punto concreto. Para además poder observar el curso de la evolución tendríamos que ver a la vez datos procedentes de varias generaciones. Esta forma de visualización puede conseguirse de varias formas. Si somos capaces de representar de forma diferente individuos procedentes de varias generaciones, entonces podremos situarlos todos juntos en un mismo gráfico. Si lo anterior no es posible se ha de recurrir a una dimensión temporal para poder seguir el proceso evolutivo. Como ejemplos de observación del estado de un población en un momento dado podemos ver cualquiera de las primeras 5 imágenes de la figura 3.1 o cualquiera de las de la figura 3.3. Como ejemplos de visualización del curso de un proceso evolutivo nos sirven los anteriores dos ejemplos si los consideramos como una secuencia de imágenes, o video, y no como imágenes aisladas. Otro ejemplo de observación del curso de la evolución, aunque resumido en un único gráfico y con menor grado de detalle sobre las soluciones individuales, son el último gráfico de

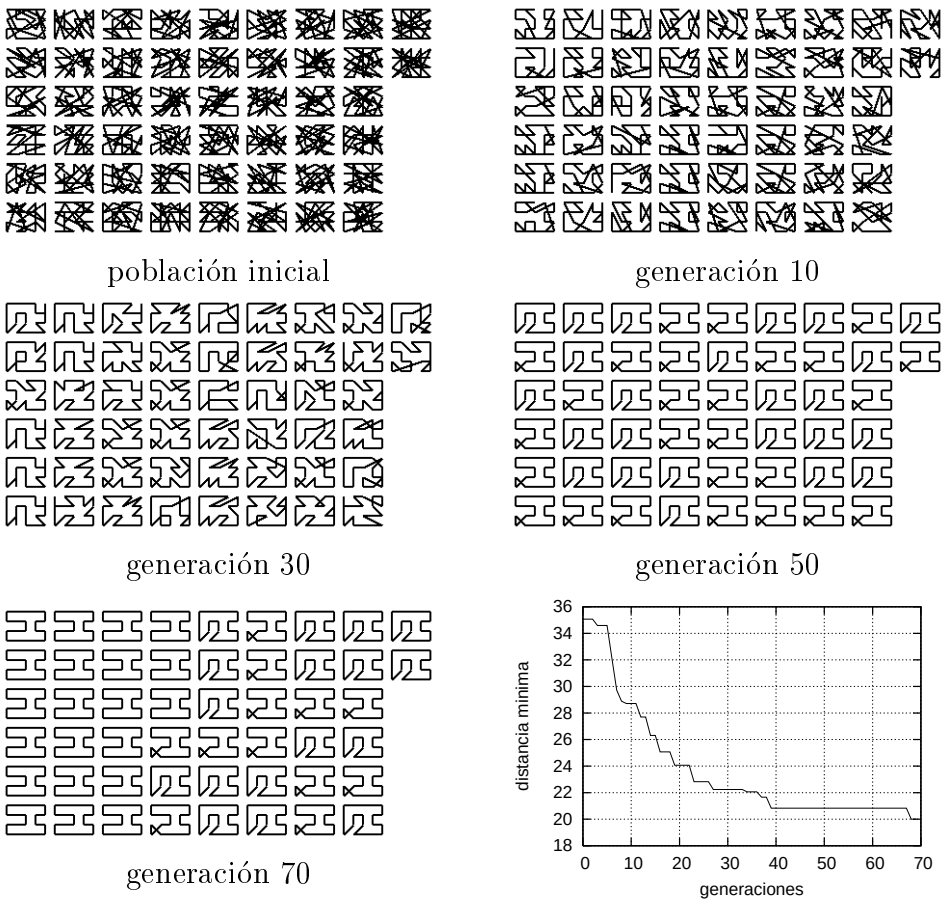


Figura 3.1: Ejemplo de resolución de un problema del viajante de comercio con 20 ciudades repartidas en una cuadrícula bidimensional de 4 filas por 5 columnas. Los 5 primeros gráficos forman una secuencia de imágenes sobre la estructura de las soluciones, es decir, el genotipo, correspondientes a varias etapas del proceso evolutivo entre el estado inicial y la generación número 70 en que se encuentra una de las soluciones óptimas existentes. El último gráfico refleja la evolución de la distancia mínima, es decir, del fenotipo, frente al número de generaciones.

la figura 3.1 y la figura 3.2b. Esta última forma es la más empleada universalmente dada su facilidad de utilización y la importancia de la información que es capaz de transmitir.

Normalmente es deseable ver la mayor cantidad de datos del mayor número de generaciones posible, siempre y cuando seamos capaces de discernir toda

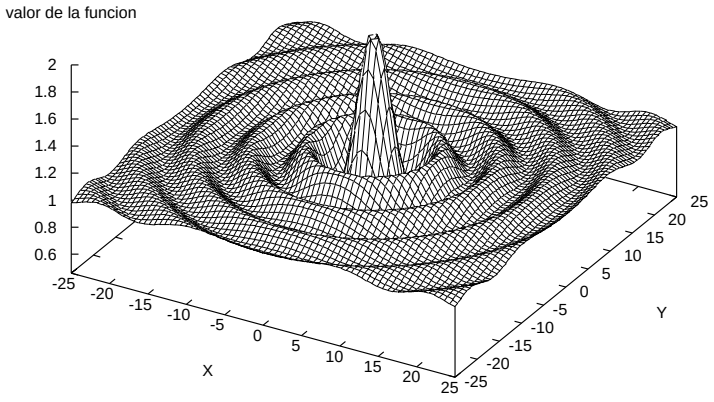
la información que se muestre. Pero como en el ejemplo del problema del viajante de comercio de la figura 3.1, en la mayor parte de las ocasiones nos vemos obligados a elegir entre cantidad de información o calidad de la misma, o, a establecer un compromiso entre ambas. En este ejemplo en concreto hemos de elegir entre visualizar los genotipos de una generación o los fenotipos de todo el proceso evolutivo. Este mismo compromiso suele darse en la inmensa mayoría de los problemas, especialmente si su estructura genética exige un representación con un elevado número de dimensiones. Si nos interesa la estructura de las soluciones normalmente nos veremos restringidos a mostrar información acerca de unas pocas generaciones, es decir, a elegir entre cantidad de información por generación o cantidad de generaciones.

En los siguientes apartados se describen detalladamente cada una de las posibilidades anteriormente mencionadas, que como veremos, a veces son excluyentes y a veces pueden ser utilizadas de forma conjunta.

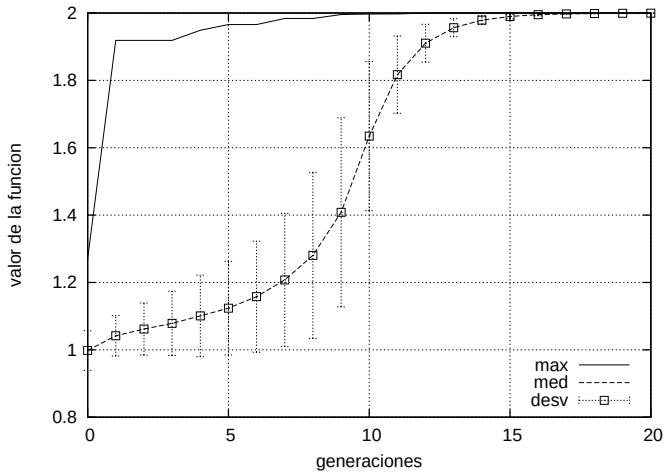
3.1.1. Genotipo y fenotipo

El genotipo es la estructura de datos que representa la solución a un problema. En ciertos problemas, como el de maximización de la función marea (Romero *et al.* , 2000), que podemos ver en la figura 3.2, puede ser algo tan simple como un par de números reales, o una una cadena de bits, que representen las dos variables x e y de la función. Para el Problema del Viajante de Comercio ("Travelling Salesman Problem" o TSP) (Lawer *et al.* , 1985), del que podemos ver un ejemplo en la figura 3.1, el genotipo puede ser un vector de números enteros que represente la ruta de ciudades a recorrer. En el caso de un problema de Programación Genética ("Genetic Programming") (Koza, 1992) puede ser un árbol que represente un algoritmo en cualquier lenguaje de programación.

El fenotipo es la manifestación del genotipo. También se le suele llamar adaptación (del inglés "fitness"). Este término biológico, en el contexto de la CE, se refiere a cómo de bueno es un individuo. Esta mediad de calidad posteriormente suele afectar a la comparación de los individuos entre sí y a la



(a) Gráfico de la función marea



(b) Ejemplo de evolución

Figura 3.2: (a) Gráfico de la función marea: $1+sin(\sqrt{x^2+y^2})/\sqrt{x^2+y^2}$. (b) Típico gráfico de la evolución del valor de la función objetivo frente al tiempo, se muestra los valores máximo, medio y la desviación estándar obtenida en cada generación.

cantidad de descendientes que un individuo podrá tener. El fenotipo se induce del genotipo y además puede tener en cuenta otros factores ambientales.

A veces es posible utilizar como fenotipo algo tan simple como un valor numérico, como en el caso de los problemas marea (sección A.1) o TSP (sección A.3). En estos casos su visualización es muy sencilla ya que puede representarse en gráficas que muestre la calidad de las soluciones frente al tiempo, o aun mejor, ser dibujados sobre un plano para ver como mejoran. Por desgracia esto no es siempre posible ya que pueden existir fenotipos mucho menos manejables, especialmente en problemas multidimensionales o en los que se busque un conjunto de soluciones óptimas, como el caso de la función F2 de Schaffer (ver sección A.4) o en el entrenamiento de redes neuronales (ver sección 5.1). En general, tanto el fenotipo como el genotipo pueden adoptar cualquier forma que un programador pueda concebir. Como ejemplo cabe destacar la biblioteca de computación evolutiva EO (Keijzer *et al.* , 2001), que es capaz de emplear como objeto de la evolución cualquier estructura de datos. Empleando los mismos ejemplos vistos anteriormente, en la maximización de la función marea, el fenotipo puede ser el valor de sustituir en la función los valores de x e y representados en el genotipo del individuo. Para el TSP puede ser la suma de las distancias entre ciudades del recorrido representado por su genotipo. Finalmente, en el ejemplo de programación genética, puede ser el grado de eficacia o de parecido con el algoritmo objetivo que se deduzca del algoritmo representado en su genotipo. En el caso de la función F2 de Schaffer el fenotipo está compuesto por dos valores reales.

En resumen podríamos decir que a través de la representación del genotipo podemos dilucidar la estructura de las soluciones y que mediante el fenotipo podemos comparar la calidad correspondiente a dichas soluciones.

3.1.1.1. Visualización del fenotipo

La visualización de los fenotipos es la técnica más empleada dentro del campo de la CE. El fenotipo de un individuo, en algunos casos sencillos, puede reducirse a un único valor numérico. Los usuarios de AE, para seguir su evolución, lo que suelen hacer es dibujar en un gráfico bidimensional, en el que en el eje X representa el tiempo, el número de generaciones transcurri-

das, o bien, el número de veces que se ha calculado dicho valor, y en el eje Y los valores del mejor individuo y/o el promedio de la población. Dos ejemplos de esta técnica quedan reflejados en el último gráfico de la figura 3.1 o en la figura 3.2b. Ciertos tipos de problemas requieren del uso de fenotipos más complejos, como, por ejemplo, los multiobjetivo (Osyczka, 1985; Goldberg & Richardson, 1987; Coello, 2000). Es este caso, y hasta ahora, se habían adoptado principalmente dos estrategias: dibujar por separado cada subobjetivo (Schaffer, 1985) o una combinación lineal de todos ellos (Syswerda & Palmucci, 1991; Jakob *et al.* , 1992). Con el método aquí propuesto se pretende poder visualizarlos todos a la vez a través del uso de una técnica de escalado multidimensional, los mapas autoorganizativos de Kohonen (Kohonen, 1990).

Esta forma de observar el progreso del AE es común a la mayoría de los problemas, ya que en la mayor parte de ellos el fenotipo puede reducirse a un único valor numérico.

3.1.1.2. Visualización del genotipo

Esta es una forma menos común de seguir la evolución de un AE. Normalmente al usar AEs sólo se inspecciona la estructura de los mejores individuos obtenidos y únicamente al final del proceso. Como ejemplo, en la búsqueda del máximo de una función de 2 variables, la función marea (ver figura 3.2), podemos dibujar dichas variables sobre un plano 2D e incluso poner cada punto sobre el gráfico 3D de la función, como podemos ver en la figura 3.3. Otro ejemplo de este tipo de visualización lo tenemos en la figura 3.1 donde la secuencia de imágenes permite ver como están hechas las soluciones del problema del viajante de comercio en varias poblaciones correspondientes a varias generaciones de la evolución de un algoritmo genético que busca caminos de longitud mínima. La ruta que representa cada individuo está dibujada por separado para cada uno de ellos. De esta forma sus longitudes pueden ser fácilmente comparadas. En las primeras etapas se utilizan rutas entre ciudades distantes y muchas de ellas son diagonales. En las últimas etapas, cuando estamos acercándonos a la solución óptima, sólo se

utilizan los caminos horizontales y verticales, los más cortos, y sólo entre ciudades muy próximas.

Desafortunadamente, esta forma de visualización depende de la representación interna de cada problema, y como dicha representación suele ser diferente no hay una forma genérica de llevarla a cabo. Así, para cada tipo de genotipo será necesario programar un método de visualización diferente. A pesar de lo anterior existen ciertos tipos de representación que son empleados muy frecuentemente, como los vectores, de forma que nos centraremos en ellos para que nuestros métodos de visualización puedan ser útiles a la mayor cantidad de usuarios de AE posible. En cualquier caso, cualquier estructura de datos puede convertirse en un vector aunque se pierda algo de información, especialmente la interrelación entre los diversos componentes de dicha estructura de datos.

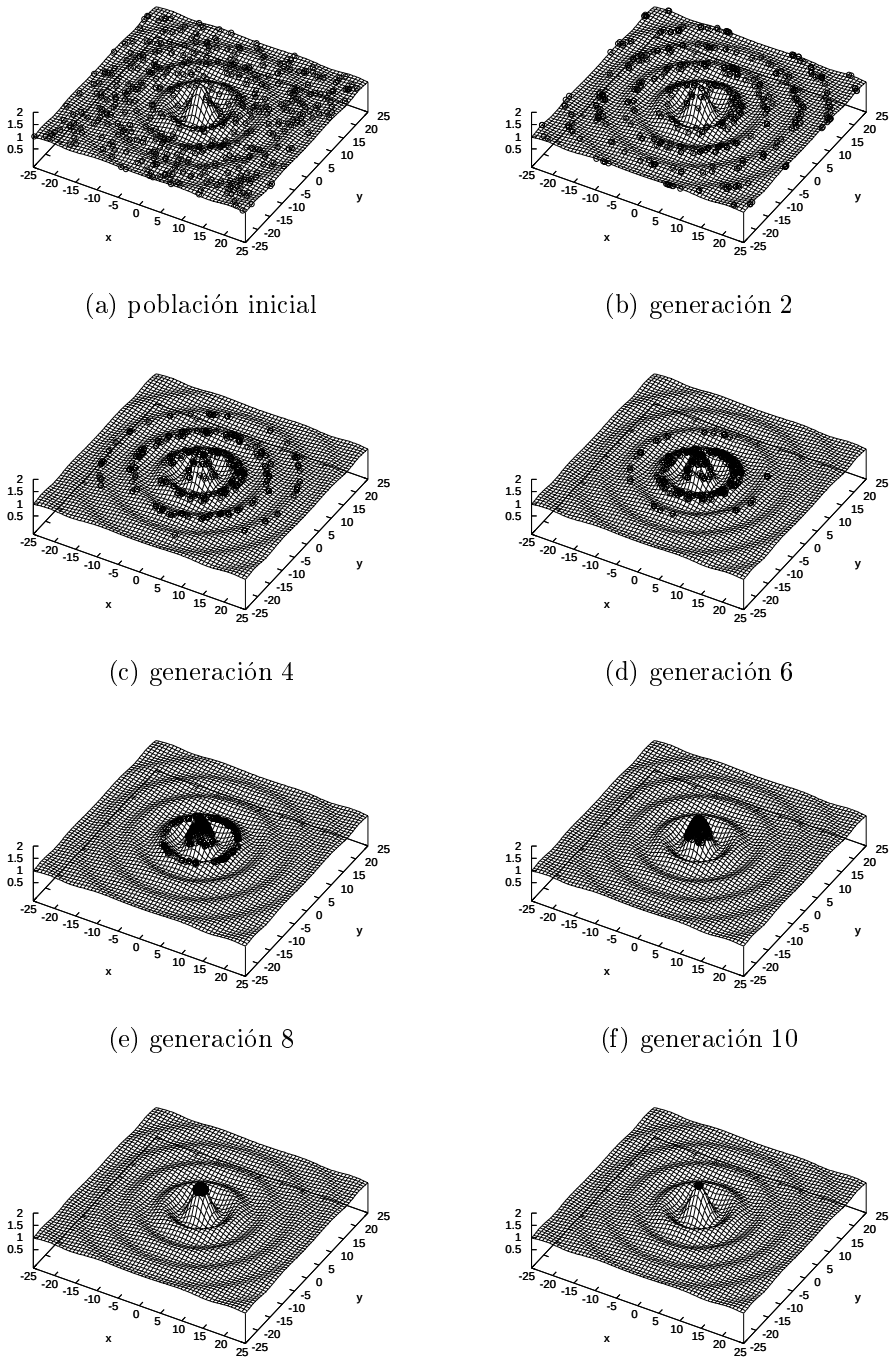
3.1.2. El estado y el curso de la evolución

Esta segunda forma de clasificación se basa en el número de generaciones de las que somos capaces de visualizar datos a la vez. Como ya hemos explicado, los datos de una generación nos bastan para hacernos una idea del estado de una población en un punto dado de su evolución. En cambio, si lo que queremos es observar como va cambiando dicha población habremos de tomar información de distintas generaciones a la vez.

Lo ideal sería poder ver datos de todo el proceso evolutivo y a la vez poder distinguir tanto las generaciones de las que proceden dichos datos, como los individuos que los generan. En general esto no es posible y hay que establecer un compromiso respecto a cual de los dos aspectos es mas importante: poder discernir a qué individuos pertenecen los datos o de qué generaciones proceden los mismos.

3.1.2.1. Visualizando el estado de la evolución

Para obtener información acerca del estado de la evolución de un AE sólo es necesaria información de la generación sobre la que estemos interesados.



(a) población inicial (b) generación 2

(c) generación 4 (d) generación 6

(e) generación 8 (f) generación 10

(g) generación 12 (h) generación 16

Figura 3.3: Secuencia de genotipos de obtenidos durante la ejecución de un algoritmo genético que busca el máximo de la función marea. En la figura 3.2 podemos ver la evolución del fitness asociada a esta misma ejecución del algoritmo.

Varias conclusiones pueden obtenerse al ver el estado de una población en un momento dado. ¿Qué áreas del espacio de búsqueda están siendo exploradas? ¿Qué grado de diversidad existe? Cualquiera de las imágenes de la figura 3.3 o las primeras 5 de la figura 3.1 pueden responder a estas importantes preguntas de un simple vistazo, la primera de ellas nos da solución a estas preguntas acerca de la optimización de la función marea, y la segunda sobre el problema TSP.

La principal ventaja de esta forma de visualizar datos es que dado que provienen todos de una misma generación podemos representarlos con mayor cantidad de detalle que si proviniesen de más. De esta forma podemos prestar mayor atención a la representación de las soluciones de forma individual, cosa que sería más difícil de hacer de tener que diferenciar además entre individuos provenientes de diferentes generaciones.

Como contrapartida cabe mencionar que a pesar de poder obtener pistas acerca del estado de una población en un instante dado, no podemos saber nada acerca de como se está desarrollando el proceso evolutivo, a no ser que encadenemos una serie de imágenes o renunciemos a representar información sobre el espacio de búsqueda, lo que nos lleva al siguiente apartado.

3.1.2.2. Visualizando el curso de la evolución

A diferencia de cuando nos interesamos por el estado de la evolución, cuando queremos averiguar algo acerca del rumbo que sigue un proceso evolutivo hemos de utilizar información procedente de varias generaciones. Como la cantidad de información que se intenta visualizar a la vez es mayor, también lo es la dificultad del proceso, de forma que habremos de establecer un compromiso entre cantidad de información y calidad de información. Cuando sólo intentábamos ver el estado del algoritmo en un punto, podía ser posible ver hasta el genotipo de cada individuo. Ahora que hemos de mezclar varias generaciones la visualización conjunta de genotipos y fenotipos puede hacerse más difícil de conseguir o llegar a ser imposible.

Un defecto de los Algoritmos Evolutivos es su forma de funcionar como un mecanismo de caja negra. Ellos realizan su trabajo, pero, en muchas

ocasiones, las personas que los utilizan no saben por qué ni cómo funcionan internamente. Si fuésemos capaces de ver cómo evoluciona una población podríamos modificar sus componentes o alterar sus parámetros para hacerlo funcionar mejor. Sería interesante determinar cómo influye cada operador genético y sus parámetros sobre una población, y cómo la afecta en cuanto a la creación de nuevas soluciones (ECVW, 1999).

Al igual que cuando describíamos la visualización de fenotipos (sección 3.1.1.1), esta es la forma de visualización más empleada, por facilidad, cuando únicamente introducimos en un gráfico estadísticas acerca de la evolución de la calidad de las soluciones a través del tiempo, como en la figura 3.2b y el último gráfico de la figura 3.1. Si por el contrario también queremos información acerca de la estructura de las soluciones normalmente ya no podremos restringirnos a un único gráfico y necesitaremos una secuencia de ellos, como en la figura 3.3 y las 5 primeras imágenes de la figura 3.1. Como antes apuntamos, hemos de elegir entre el nivel de detalle de la información o la cantidad de generaciones de las que mostrar información. Los dos primeros ejemplos muestran información sobre todo el proceso evolutivo en conjunto y en una única imagen, aunque con un nivel de detalle escaso, sólo el fitness. Por el contrario, los dos últimos ejemplos, necesita de varias imágenes para mostrarnos información acerca de todo el proceso, pero con un nivel de detalle mucho mayor, dado que informan sobre la calidad y la estructura de las soluciones, así como sobre el grado de diversidad existente en la población y las áreas del espacio de búsqueda que están siendo exploradas.

3.2. Visualización multidimensional

La mayoría de las técnicas de visualización están limitadas a representar datos que dependen de no más de tres variables. Esto se debe al hecho de que la visión humana está restringida a tres dimensiones. Hay varias formas de sobrepasar esta limitación: una es usar el color como cuarta dimensión, y otra, usar el tiempo como la quinta. Ninguna de esta posibilidades es una solución ideal y además puede requerir cierta práctica el llegar a comprender

lo que se está mostrando. De todas formas, existen multitud de problemas que superan con creces esta limitación de cinco variables, con lo cual estos trucos, además de poco adecuados, pueden resultar insuficientes.

La solución más empleada es la de utilizar algún método que nos permita reducir el número de dimensiones de un problema de cualquier número arbitrariamente alto, n, a 2 ó 3. Dicho método debe proporcionar una imagen en 2 ó 3 dimensiones de lo que sucede en el espacio n-dimensional original. Para ello las relaciones existentes entre los puntos en el espacio multidimensional deben mantenerse tanto como sea posible en el espacio con menor número de dimensiones. A estos métodos se les denomina de *escalado multidimensional* o de *proyección* (Cox & Cox, 1994; Ripley, 1996; Tsogo & Masson, 2000).

Para medir el parecido o diferencia entre los puntos n-dimensionales suele utilizarse algún tipo de distancia. Estas distancias se toman en el espacio con mayor número de dimensiones y después se comparan con las distancias en el espacio con un menor número de dimensiones para medir la calidad del escalado multidimensional. Dos de los tipos de distancia más frecuentemente empleadas son la distancia euclídea (ecuación 3.1) y la distancia de Manhattan , denominada de Hamming cuando los componentes son binarios, (ecuación 3.2).

$$d_E(v_1, v_2) = \sqrt{\sum_{i=1}^{n} (v_{1i} - v_{2i})^2} \qquad v_1, v_2 \in \mathbb{R}^n \tag{3.1}$$

$$d_M(v_1, v_2) = \sum_{i=1}^{n} (v_{1i} - v_{2i}) \qquad v_1, v_2 \in \mathbb{R}^n \tag{3.2}$$

Existen multitud de métodos de proyección, cada uno de ellos con diferentes virtudes y defectos. König, en (König, 1998), hace un revisión de las técnicas de proyección no supervisadas prestando especial atención a su complejidad computacional y a la preservación de las estructuras n-dimensionales. Arthur Flexer en (Flexer, 1999) y Juha Vesanto en (Vesanto, 1999) llevan a cabo dos estudios sobre las propiedades del SOM cuando se utiliza para la visualización datos multidimensionales. Algunas de las técnicas más conocidas

son:

- Análisis de Componentes Principales (PCA)
- Escalado multidimensional (MDS)
- Algoritmo de Sammon
- Algoritmo k-medias
- Mapa Auto-Organizativo de Kohonen (SOM)
- Análisis de Componentes Curvilíneos (CCA)
- Análisis de Distancias Curvilíneas (CDA)
- Locally Linear Embedding (LLE)

De los anteriores métodos, los lineales como el PCA, son fáciles y rápidos de calcular, pero no permiten proyectar estructuras no lineales de forma correcta. Los métodos no lineales, como el de Sammon, pueden proyectar estructuras no lineales pero su coste computacional es mucho más elevado. Algunas de las técnicas enumeradas, como el CDA o el SOM, permiten incluso controlar si se desean primar las distancias locales frente a las globales, o viceversa, durante el proceso de proyección. A continuación se examinan con más detalle cada una de estas técnicas.

3.2.1. Análisis de Componentes Principales (PCA)

El análisis de componentes principales (PCA)(Hotelling, 1933) es un procedimiento matemático que transforma un conjunto de variables posiblemente correlacionadas en otro conjunto menor de variables no correlacionadas a las que se denomina componentes principales. El primer componente principal recoge tanta variabilidad del conjunto de datos como sea posible, y cada componente sucesivo recoge tanta variabilidad de la restante como sea

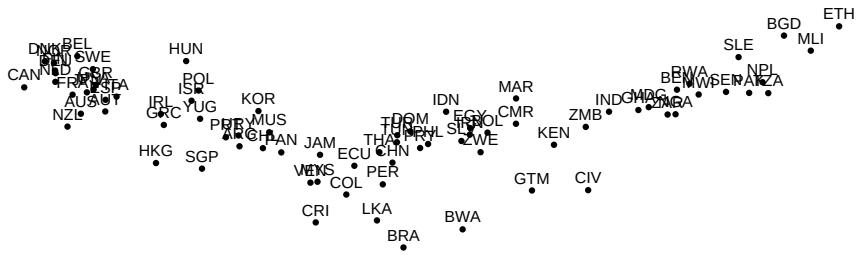


Figura 3.4: Ejemplo de proyección multidimensional empleando el Análisis de Componentes Principales (PCA). El conjunto de datos representa 39 aspectos acerca de la riqueza o pobreza referentes a 77 países.

posible. Los objetivos de ésta técnica son dos: descubrir o reducir la dimensionalidad de un conjunto de datos e identificar cuan significativas son las variables subyacentes.

En la figura 3.4 se puede observar la proyección de un conjunto de datos multidimensionales sobre el plano utilizando el PCA. El conjunto de datos está compuesto por vectores de 39 componentes que describen diferentes aspectos de la riqueza o pobreza de 77 países. Esto datos se tomaron de un informe sobre el desarrollo del Mundo en 1992. Para más detalles ver (Kaski, 1997).

3.2.2. Escalado multidimensional (MDS)

Este es un grupo de métodos ampliamente utilizado en ciencias económicas y sociales para analizar evaluaciones subjetivas de entidades tales como encuestas o estudios de mercado. Las primeras versiones de MDS para datos métricos fueron desarrolladas en los años 30. Este algoritmo fue diseñado para analizar una matriz de disimilitudes, y por tanto, podía usarse para reducir la dimensionalidad de un conjunto de datos. Estos métodos pueden dividirse en dos grandes categorías:

- **Metric MDS**: En la versión original de Torgerson (Trogerson, 1952) se requerían las distancias entre los puntos del conjunto de datos y otra configuración de puntos en un espacio euclídeo a la que se deseaba

asemejarlos, a menudo una proyección lineal en un subespacio obtenida a través de PCA. La idea del método era aproximar el conjunto original de distancias a la configuración en el espacio euclídeo, lo cual permite construir un método de proyección no lineal. Para ello a cada elemento $x_k^n \in \mathbb{R}^n$ del conjunto original se representa con otro de menor cardinalidad $x_k^p \in \mathbb{R}^p$. El objetivo del algoritmo es optimizar la representación de manera que las distancias entre los puntos del conjunto con menor número de dimensiones sean lo más parecidas posibles a las de los elementos originales. Empleando una función de error cuadrática y representando como d_{ij}^n a la distancia entre los elementos $x_i^n, x_j^n \in \mathbb{R}^n$ y d_{ij}^p a la distancia entre los elementos $x_i^p, x_j^p \in \mathbb{R}^p$, la función objetivo a minimizar puede escribirse como:

$$E_{mMDS} = \sum_{i=0}^{N} \sum_{j=0}^{N} (d_{i,j}^n - d_{i,j}^p)^2$$

- **Nonmetric MDS**: La reproducción de las distancias euclídeas no es siempre la mejor opción, especialmente cuando los componentes del conjunto de datos se ordenan mediante una escala ordinal. En este caso sólo el orden de rango es significativo, y no el valor exacto de dicho número. En este caso la mejor opción es emplear una función monótonamente creciente en función de las distancias originales que preserve el orden establecido. De esta forma Kruskal (Kruskal, 1964) y Shepard (Shepard, 1962) proponen dos métodos de este tipo basados en funciones así normalizadas, f, quedando la función de error a minimizar así:

$$E_{nmMDS} = \frac{1}{\sum_{i=0}^{N} \sum_{j=0}^{N} (d_{i,j}^p)^2} \sum_{i=0}^{N} \sum_{j=0}^{N} (f(d_{i,j}^n) - d_{i,j}^p)^2$$

En la figura 3.5 se puede ver un ejemplo del uso de MDS no métrico para proyectar el mismo conjunto de datos que podemos ver proyectado en la figura 3.4 mediante el PCA.

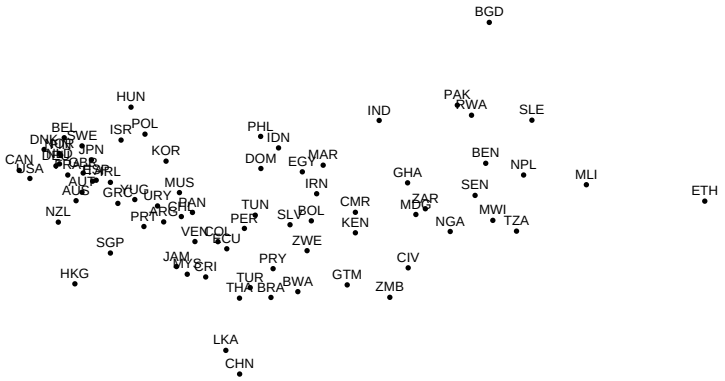


Figura 3.5: Ejemplo de proyección multidimensional empleando el Análisis de Componentes Principales (PCA). El conjunto de datos representa 39 aspectos acerca de la riqueza o pobreza referentes a 77 países.

3.2.3. Algoritmo de Sammon

Este es uno de los algoritmos más empleados en escalado multidimensional. Fue propuesto por Sammon en (Sammon Jr., 1969). Lo que hace es tomar un conjunto de datos n-dimensionales y devolver otro conjunto de datos con un número menor de dimensiones, por ejemplo, p intentando conservar las relaciones de distancia existentes entre los datos originales. Realmente es una variante más de MDS cuya fórmula de error a minimizar es:

$$E_{Sammon} = \sum_{i=0}^{N} \sum_{j=0}^{N} \frac{(d_{i,j}^{n} - d_{i,j}^{p})^2}{d_{i,j}^{n}} = \frac{1}{\sum_{i=0}^{N} \sum_{j=0}^{N} (d_{i,j}^{n})^2} E_{mMDS}$$

donde $d_{i,j}^{n}$ y $d_{i,j}^{n}$ son las distancias entre los vectores i y j del espacio n-dimensional y p-dimensional, respectivamente.

En la figura 3.6 se puede ver un ejemplo del uso del algoritmo de Sammon para proyectar el mismo conjunto de datos sobre el reparto de la riqueza en el mundo cuyas proyecciones usando PCA y MDS ya se han visto en las figuras 3.4 y 3.5.

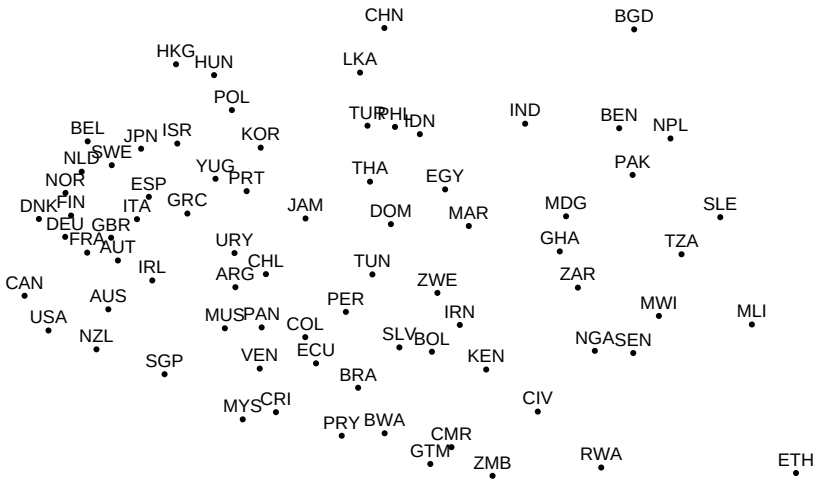


Figura 3.6: Ejemplo de proyección multidimensional empleando el Análisis de Componentes Principales (PCA). El conjunto de datos representa 39 aspectos acerca de la riqueza o pobreza referentes a 77 países.

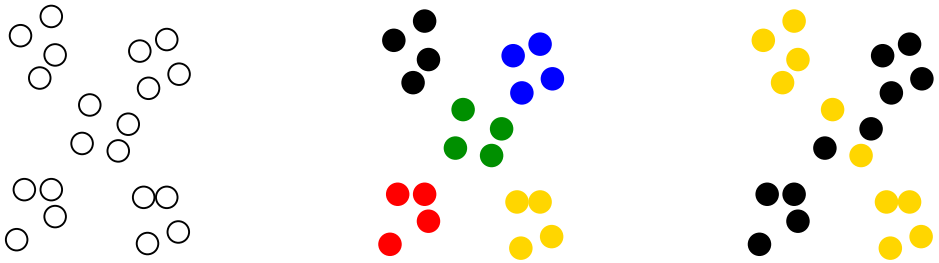
3.2.4. Algoritmo k-medias

Este algoritmo, muy utilizado en minería de datos, también es aplicable en visualización. A veces se utiliza en conjunción con el algoritmo de Sammon como hace Flexer en (Flexer, 1997) y (Flexer, 1999), y otras, por sí solo tal y como describe Modha en (Modha *et al.* , 1998).

Flexer, en su método oKMC+, busca un conjunto de vectores que produzcan un partición de distorsión mínima mediante el algoritmo k-medias y a continuación le aplica el algoritmo de Sammon para poder visualizar un conjunto de datos.

Modha en cambio lo que hace es elegir dos centroides, cuyas distancias a cada vector a proyectar utiliza como coordenadas x e y para dibujarlos sobre el plano. Lo que hace realmente útil a este método es que también propone una técnica para moverse suavemente entre proyecciones, y, de esta forma, se pueden enlazar varias proyecciones como si de una secuencia de video se tratase.

El único parámetro que utiliza este algoritmo, k, indica el número de



(a) Conjunto de datos (b) $k = 5$ (c) $k = 2$

Figura 3.7: Ejemplo de uso del algoritmo k-medias para clasificar un conjunto de datos en diferentes subconjuntos. (a) Conjunto de datos. (b) Clasificación para $k = 5$. (c) Clasificación para $k = 2$.

grupos en los que se dividirá el conjunto de datos de entrada. El la figura 3.7 podemos ver un conjunto de datos y dos clasificaciones de los mismos empleando dos valores diferentes de k.

3.2.5. Mapa Auto-Organizativo de Kohonen (SOM)

El mapa auto-organizativo de Kohonen (del inglés "Self Organizing Map" o "SOM") es un tipo de red neuronal artificial descrita por Teuvo Kohonen (Kohonen, 1990; Kohonen, 1997; Kohonen, 1998). Su algoritmo de entrenamiento es del tipo competitivo y no supervisado. Habitualmente es empleado en la cuantización y proyección de vectores, así como en visualización. Su función de error es:

$$E_{SOM} = \sum_{i=0}^{N} \sum_{k=0}^{K} (d_{i,k}^{n})^2 h_{ck}(d_{i,j}^{p})$$

donde K es el conjunto de datos de entrenamiento y h_{ck} es una función de vecindad centrada en la neurona c y monótonamente decreciente en función del tiempo.

En su forma más simple, la originalmente descrita por Kohonen, está compuesto por un conjunto de neuronas dispuestas como un vector de 1 ó 2 dimensiones. Aunque pueden utilizarse más dimensiones, no es habitual porque

sería más difícil de visualizar. Entre las neuronas se define una función de vecindad, habitualmente de forma hexagonal o rectangular. Cada neurona posee un vector n-dimensional m_i. El algoritmo de entrenamiento es iterativo. En cada paso del algoritmo de entrenamiento repite el siguiente proceso:

1. Escoger un vector x del conjunto de entrenamiento.

2. Calcular la distancia entre x y cada uno de los vectores m_i de las neuronas. A la neurona más cercana a x, escogida mediante la ecuación 3.3, se la denomina neurona ganadora.

$$\|x - m_g\| = \min_i \{\|x - m_i\|\} \tag{3.3}$$

3. Actualizar los vectores m_g de la neurona ganadora y m_i de sus vecinas más cercanas según la ecuación 3.4

$$m_i(t+1) = m_i(t) + \alpha(t) h_{gi}(t)[x - m_i(t)] \tag{3.4}$$

   $\alpha(t)$ es la tasa de aprendizaje y es $h_{gi}(t)$ la función de vecindad centrada sobre la unidad ganadora. Ambas son funciones monótonamente decrecientes en función del tiempo.

En la figura 3.8 se puede ver la proyección, empleando el SOM, del mismo conjunto de datos sobre reparto de la riqueza en el mundo empleado como ejemplo del uso de otras técnicas las figuras 3.4, 3.5 y 3.6.

En visualización este método es especialmente apreciado por crear una función que preserva la topología del espacio con gran número de dimensiones, en el espacio con un menor número de ellas, es decir, mantiene las relaciones de distancia existente en el espacio original, en otro de menor cardinalidad. Así, los puntos cercanos en el espacio original también lo estarán en el transformado, y viceversa, los lejanos también lo seguirán estando. Además posee la propiedad de generalizar. Esto posibilita que podamos presentar a la red nuevos puntos con los que no ha sido entrenada y que sean correctamente

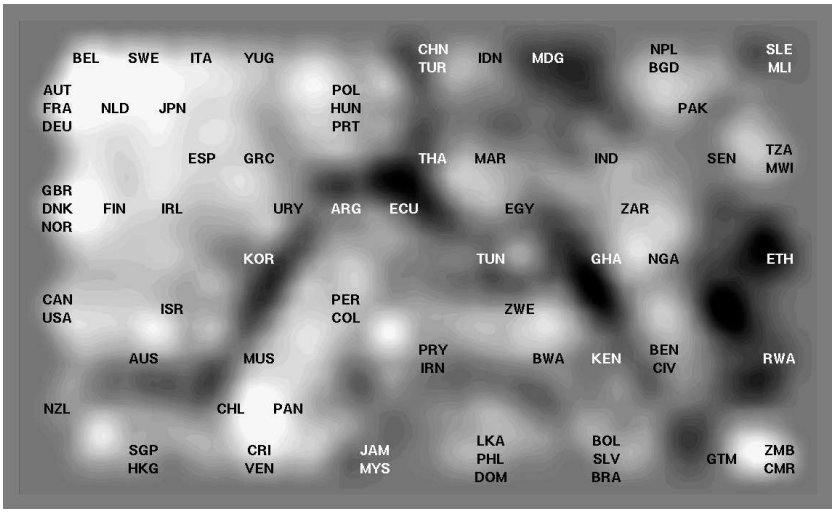


Figura 3.8: Ejemplo de proyección multidimensional empleando el SOM. El conjunto de datos representa 39 aspectos acerca de la riqueza o pobreza referentes a 77 países.

clasificados y/o proyectados. Otros métodos de escalado multidimensional en cambio deben ser reentrenados cada vez que deseamos utilizar nuevos datos.

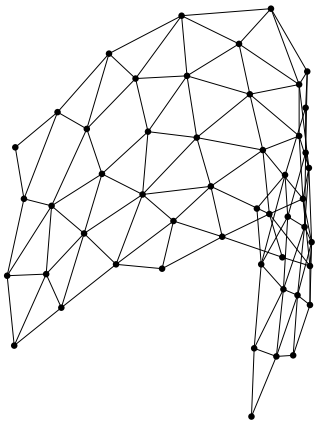
En algunos casos, cuando el número de dimensiones del espacio de entrada es pequeño, otros métodos como oKMC+, CCA o CDA pueden proporcionar proyecciones con menor error de cuantización o preservar mejor la topología global, pero a medida que el número de dimensiones crece el comportamiento del SOM es mejor que el de estos otros métodos (Flexer, 1999; Vesanto, 1999).

Otra de las ventajas del SOM es que permite elegir donde colocar ciertos puntos fijos, es decir, nos permite elegir en que neuronas se proyectarán ciertas entradas. De esta forma podemos hacer que determinados tipos de individuos o áreas de interés sean proyectadas siempre donde nos convenga, incluso a lo largo de diferentes mapas.

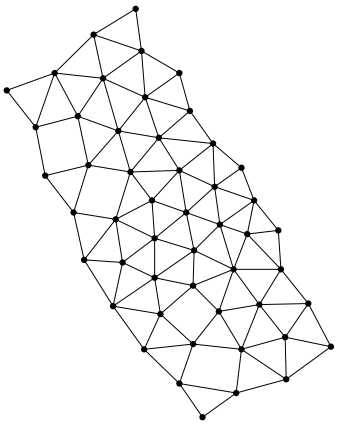
3.2.6. Análisis de Componentes Curvilíneos (CCA)

CCA es una red neuronal auto-organizativa que proporciona un mapeado en baja dimensionalidad de un subespacio de un conjunto de datos no lineal

de alta dimensionalidad. Se basa en la creación de una forma de relacionar un espacio de entrada con otro de salida a través de un conjunto de neuronas, cada una de las cuales posee dos conjuntos de vectores: uno para la entrada y otro para la salida. En la figura 3.9 podemos ver un ejemplo de la estructura de este tipo de red neuronal.



(a) Neuronas de entrada (3D)



(b) Neuronas de salida (2D)

(c) Conjunto de datos (3D)

(d) Proyección (2D)

Figura 3.9: Estructura de la red neuronal auto-organizativa interna empleada en el Análisis de Componentes Curvilíneos (CCA) de Pierre Demartines para la proyección de un conjunto de datos con forma de casco de caballo. (a) Vectores de entrada de la red neuronal. (b) Vectores de salida de la red neuronal. (c) Conjunto de datos 3D que se desea proyectar. (d) Proyección 2D del conjunto de datos.

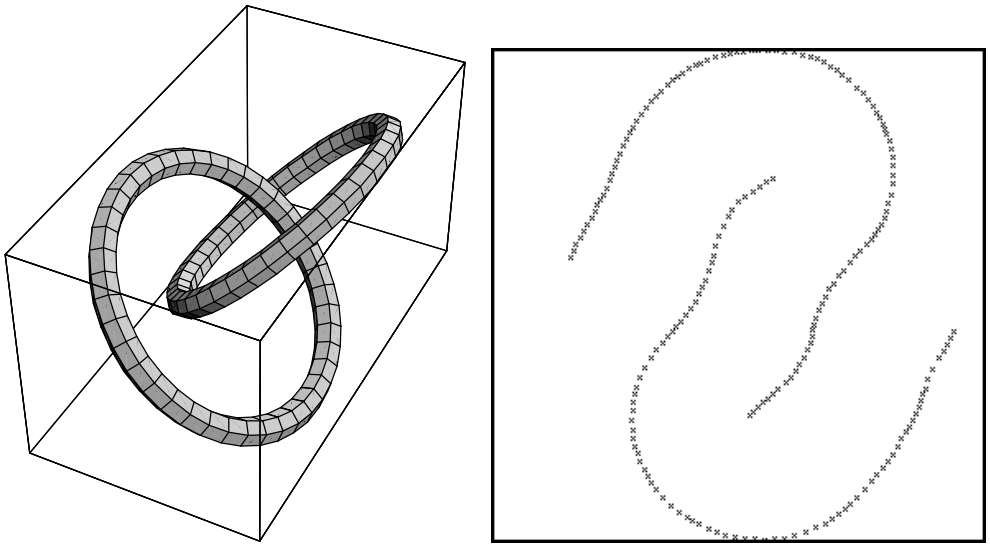
Este algoritmo fue propuesto por Pierre Demartines (Demartines & Hérault, 1997) como una mejora del SOM y es muy similar en sus objetivos a otros métodos no lineales tales como el MDS o el algoritmo de Sammon. Su salida no tiene un tamaño fijo sino que representa un espacio continuo capaz de tomar cualquier forma. Para utilizarlo se han de llevar a cabo dos procesos: uno de cuantización vectorial y otro de proyección no lineal. La parte de proyección es similar a otros métodos como el MDS o el algoritmo de Sammon. La función de error que minimiza tiene la siguiente forma:

$$E_{CCA} = \sum_{i=0}^{N} \sum_{j=0}^{N} (d_{i,j}^{n} - d_{i,j}^{p})^2 F(d_{i,j}^{p})$$

Donde $F(d_{i,j}^{p})$ es una función acotada y monótonamente decreciente del espacio de salida, cuyo objetivo es es favorecer la conservación de las relaciones topológicas locales frente a las globales. Al igual que el SOM tiene la capacidad de proyectar nuevos puntos desde el espacio original al del dimensionalidad reducida una vez entrenada la red neuronal. El CCA tiene la ventaja de poseer un menor número de parámetros, pero en contra, como se constata en (Lendasse *et al.* , 2000), tiene la desventaja de ser más impreciso.

3.2.7. Análisis de Distancias Curvilíneas (CDA)

CDA es una versión mejorada del CCA de Demartines (Demartines & Hérault, 1997) propuesta por John Aldo Lee (Lee *et al.* , 2000; Lee *et al.* , 2002) con un mejor comportamiento en la proyección de conjuntos de datos con relaciones altamente no lineales y con una completa automatización en el proceso de elección de valores para los parámetros. La función de error permanece prácticamente idéntica salvo por la utilización de la distancia curvilínea $\delta_{i,j}^{n}$ en lugar de la euclídea $d_{i,j}^{n}$ para calcular las diferencias entre los puntos del espacio original. En la figura 3.11 se puede ver la diferencia entre ambos tipos de distancia. La función de error ahora tiene este aspecto:



(a) Anillos entrelazados en 3D (b) Proyección 2D de los anillos

Figura 3.10: Ejemplo de uso del Análisis de Componentes Curvilíneos (CCA) de Pierre Demartines para proyectar dos anillos tridimensionales y enlazados sobre el plano.

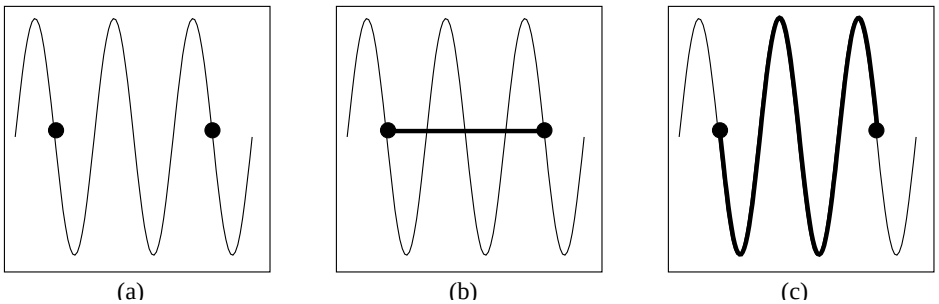
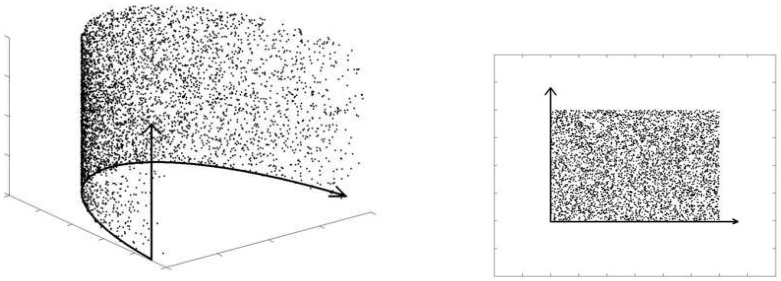


Figura 3.11: Diferencia entre las distancias euclídea y curvilínea: (a) dos puntos sobre una estructura curvilínea, (b) distancia euclídea entre los dos puntos, (c) distancia curvilínea entre los dos puntos.



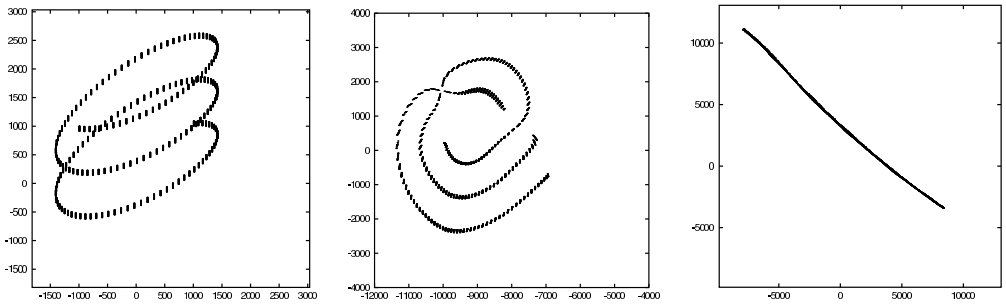
(a) Conjunto de datos 3D (b) Proyección 2D

Figura 3.12: Ejemplo de uso del Análisis de Distancias Curvilíneas (CDA) de John Aldo Lee para proyectar un conjunto de datos con forma de casco de caballo de 3D a 2D. (a) Conjunto de datos original en 3D. (b) Proyección 2D del conjunto de datos.

$$E_{CDA} = \sum_{i=0}^{N} \sum_{j=0}^{N} (\delta_{i,j}^{n} - d_{i,j}^{p})^2 F(d_{i,j}^{p})$$

Este método presenta dos ventajas frente al CCA: un mejor comportamiento para conjuntos de datos altamente no lineales, como espirales, y una elección de parámetros completamente automática. El CDA hereda el CCA un menor precisión en comparación con el SOM, aunque como ventaja frente a este presenta un algoritmo de entrenamiento más eficiente, $O(n)$ frente a $O(n^2)$.

En la figura 3.12 podemos ver un ejemplo de proyección de un conjunto de datos con forma de casco de caballo de 3D a 2D. Es el mismo ejemplo que se utilizó en la figura 3.9 para demostrar el uso del CCA. En este caso, dada la linealidad del conjunto de datos, los resultados entre el CCA y el CDA son parecidos. A continuación se verá otro ejemplo en que dada la elevada no linealidad del conjunto de datos, el CDA funciona mucho mejor que el CCA. Esta vez lo que se intenta proyectar es un conjunto de datos con forma de espiral en 3 dimensiones. Los resultados, que pueden verse en la figura 3.13, muestran una clara superioridad del CDA para este tipo de conjuntos



(a) Conjunto de datos (b) Proyección CCA (c) Proyección CDA

Figura 3.13: Comparativa entre CCA y CDA en la proyección de conjuntos de datos altamente no lineales. (a) Conjunto datos con forma de espiral en 3D. (b) Proyección 2D mediante CCA. (c) Proyección 2D mediante CDA.

de datos.

3.2.8. Locally Linear Embedding (LLE)

LLE nace con el objetivo de descubrir representaciones compactas de datos de alta dimensionalidad para la exploración, análisis y visualización de los mismos. LLE es un algoritmo de aprendizaje no supervisado capaz de calcular representaciones de baja dimensionalidad que mantienen las relaciones de vecindad de unas entradas con un mayor número de dimensiones. Está basado en simples intuiciones geométricas descritas por Sam T. Roweis en (Roweis & Saul, 2000). Este método supone que cada punto y sus vecinos están cercanos en un área del espacio. Entonces lo que se intenta es describir esta área a través de un vector de coeficientes lineales, w, que reconstruyen cada punto del espacio a partir de sus vecinos. Los errores de reconstrucción se mide mediante la función de costo de la ecuación 3.5:

$$E_w = \sum_{i=0}^{N} \left| x_i - \sum_{j=0}^{N} w_{ij} x_j \right|^2 \tag{3.5}$$

Donde x_i y x_j son los datos i-ésimo y j-ésimo, respectivamente, y w_{ij} representan la contribución de cada componente del conjunto de datos x_j a la i-ésima reconstrucción.

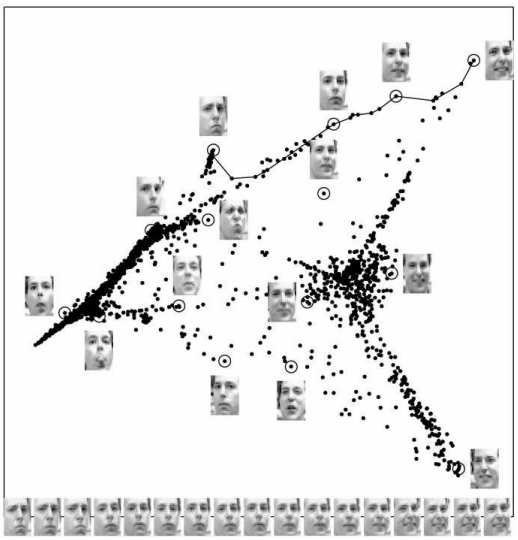


Figura 3.14: Ejemplo de uso de Locally Linear Embedding (LLE) de Sam. T. Roweis. Proyección de caras humanas. Los círculos representan la posición de las caras cercanas. La caras de la parte inferior se corresponden con los puntos a lo largo de la línea continua de la parte superior-derecha.

Se elimina la necesidad de calcular la distancia o diferencias entre cada par de puntos del conjunto de datos, centrando su atención sólo en los que están más cercanos entre sí. De aquí que este método sea bastante rápido de calcular, pero que tenga un punto débil: falla cuando el conjunto de datos está divido en varios grupos separados, según se demuestra en (Perona & Polito, 2002).

El la figura 3.14 se puede ver al LLE en acción proyectando un conjunto de caras humanas con diferentes poses y expresiones.

3.3. Visualización de espacios de búsqueda mediante el SOM

El método de visualización aquí propuesto se basa en el uso de los mapas autoorganizativos de Kohonen como técnica de escalado multidimensional para la proyección de individuos sobre la representación de un espacio de

búsqueda.

En función de nuestros intereses podemos escoger cómo entrenar los mapas de forma que nos permitan representar el espacio de búsqueda de un problema o el espacio explorado por un algoritmo evolutivo durante la búsqueda de la solución. Cuando lo que se desea es conocer la forma del espacio de búsqueda de un problema lo mejor será entrenar el SOM con una nube de puntos escogidos aleatoriamente de dicho espacio de búsqueda. En cambio para comparar diferentes ejecuciones de un mismo algoritmo evolutivo es mucho más ventajoso centrarse únicamente en las áreas exploradas, y no en todo el espacio de búsqueda. Para ello el SOM debe entrenarse con soluciones obtenidas por el algoritmo evolutivo durante su ejecución. En el capítulo 4 se hace una descripción más detallada acerca de las diferencias entre un espacio del búsqueda y el espacio explorado por un algoritmo genético durante su funcionamiento.

La utilización del SOM para visualizar un espacio de búsqueda requiere de la ejecución de dos pasos: la obtención del SOM y la proyección de individuos sobre el mismo. La obtención del SOM, a su vez, requiere llevar a cabo tres procesos:obtener el conjunto de entrenamiento, entrenar el mapa y verificar el correcto entrenamiento del mismo (ver figura 3.15).

El primero proceso que debemos realizar es la selección del conjunto de entrenamiento. De este dependerá en gran medida la capacidad de visualización que posea el SOM una vez entrenado. Lo primero que debemos decidir es que deseamos ver: un espacio de búsqueda o la porción de un espacio de búsqueda explorada por un algoritmo evolutivo durante su funcionamiento.

En el primer caso, es decir, si deseamos visualizar un espacio de búsqueda, lo normal será escoger una serie de puntos de dicho espacio de búsqueda al azar. En caso de que poseamos alguna pista acerca de la forma de este espacio búsqueda, podemos incorporarla a la forma de escoger los puntos. En la sección 4.1 se pueden encontrar dos ejemplos de selección de puntos al azar para visualizar un espacio de búsqueda y como después se aplica el conocimiento específico de la forma de dichos espacios para afinar el proceso de visualización. Esto puede llevarse a cabo variando la función de probabil-

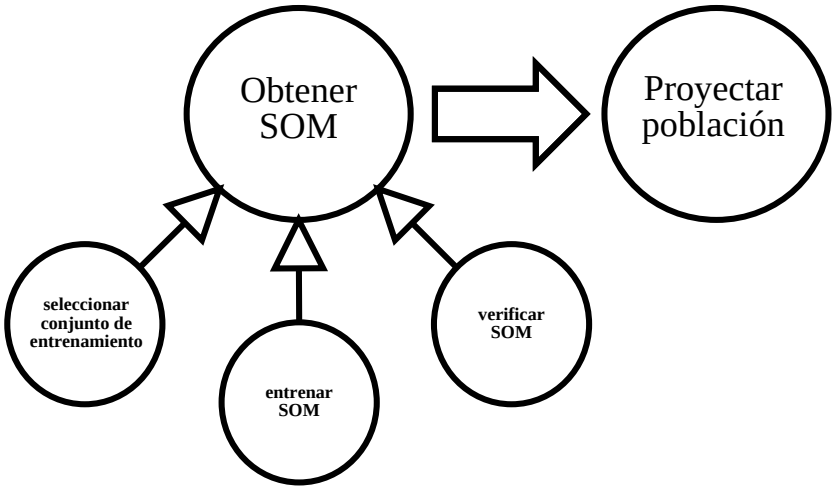


Figura 3.15: Método de proyección basado en el SOM. En primer lugar se ha de obtener el SOM sobre el que proyectar los individuos de un población. Para obtener dicho mapa es necesario escoger un conjunto de entrenamiento adecuado, entrenar el mapa de forma adecuada y comprobar que dicho proceso de entrenamiento ha producido un mapa correcto.

idad con la que se escogen los datos. De esta forma también se verá afectada la función de densidad de probabilidad del espacio de búsqueda representado. Este proceso se lleva a cabo sobre dos problemas típicos como son onemax (sección 4.1.1) y Rastrigin (sección 4.1.2).

Si por el contrario lo que se desea es visualizar el espacio explorado por un algoritmo evolutivo, deberemos crear el conjunto de entrenamiento a partir de individuos obtenidos por el propio algoritmo durante su funcionamiento. Según que deseemos poner de manifiesto podemos obtener dichos individuos de una o varias ejecuciones del algoritmo. Para poner de manifiesto las diferencias entre las fases tempranas y tardías del proceso evolutivo puede es útil emplear individuos de una única ejecución. En cambio, para comparar diferentes ejecuciones e intentar descubrir tendencias del algoritmo puede ser útil emplear individuos procedentes de varias ejecuciones. En la sección 4.2.1 hay un ejemplo de este tipo de selección del conjunto de entrenamiento aplicado al problema de la mochila. A título comparativo, en la sección 4.2.2 podemos encontrar una visualización de todo el espacio de búsqueda de este problema.

En general, y salvo para comparar varias ejecuciones de un mismo algoritmo, es más ventajoso intentar visualizar todo el espacio de búsqueda de un problema. Cuando se utilizan puntos del espacio explorado durante cierto número de ejecuciones no se garantiza que se cubra todo el espacio de búsqueda, con lo cual se pueden ignorar ciertas áreas de interés, y se corre el riesgo de centrarse únicamente en las que el algoritmo encuentra con mayor facilidad.

Una vez obtenido el conjunto del entrenamiento podemos pasar a entrenar el SOM. La implementación utilizada para ello ha sido **SOM_PAK** (Kohonen *et al.* , 1996) disponible de forma gratuita en Internet en la dirección `http://www.cis.hut.fi/research/som-research/nnrc-programs.shtml`. En este documento se describen los principios de los mapas autoorganizativos junto con los programas de que consta el paquete y se dan una serie de consejos prácticos para la creación de buenos mapas, entre los cuales cabe destacar:

- La forma del mapa debe ser preferiblemente alargada para conseguir una orientación estable de los datos a lo largo de la función de densidad de probabilidad que aproxima el mapa. Esto es especialmente importante cuando el conjunto de entrenamiento se escoge mediante individuos obtenidos durante la ejecución del algoritmo evolutivo o cuando se escogen de forma no aleatoria del espacio de búsqueda del problema. En caso de obtenerse aleatoriamente, ya no existirán direcciones más importantes que otras.

- Para obtener mapas con una buena precisión estadística es necesario repetir el proceso de entrenamiento durante una gran cantidad de pasos, como, por ejemplo, 100000 pasos. A mayor número de neuronas se requerirán más pasos para obtener mapas con la misma precisión que otros de menor tamaño.

- Las neuronas del SOM pueden disponerse siguiendo una distribución rectangular o hexagonal, siendo esta última más conveniente para su utilización en visualización.

- El mejor mapa se espera que sea el que tenga el menor error de cuantización. Dicho error puede medirse mediante un programa llamado `qerror`.

Para entrenar el SOM en primer lugar hemos de inicializar los pesos de las neuronas a partir del conjunto de datos con el que se desea entrenarlo, bien de forma aleatoria con el programa `randinit`, o bien de forma ordenada a lo largo de un subespacio bidimensional que sigue los componentes principales de dicho conjunto de datos de entrada con el programa `lininit`. Una vez inicializado puede entrenarse mediante el programa `vsom`.

Una vez entrenado sólo falta comprobar que el mapa es correcto y que su precisión es la adecuada. Comprobar la precisión es sencillo y puede hacerse midiendo su error de cuantización con el programa `qerror`. El comprobar la corrección es algo más complicado y podemos hacerlo con los siguientes programas:

- `visual`: Este programa genera una lista de coordenadas correspondientes a las neuronas ganadoras del mapa para cada muestra de un conjunto de datos. También da el error de cuantización de cada muestra y su etiqueta correspondiente, si esta ha sido definida.

- `sammon`: Genera un mapa de Sammon en 2 dimensiones a partir de un conjunto de datos n-dimensional donde los puntos tienden a aproximarse según la distancia euclídea entre ellos.

- `planes`: Este programa genera un gráfico que muestra la proyección de un cierto componente del mapa sobre un plano. Si le pasamos un conjunto de datos generará un segundo gráfico con la trayectoria formada por la neuronas ganadoras para cada muestra.

- `umat`: Este programa genera un gráfico que permite visualizar las distancias entre las neuronas vecinas de un SOM mediante niveles de grises tal como se describe en (Ultsch, 1993). A mayor diferencia entre vecinos, más oscuro será el color que los separe. Todos las proyecciones

que se verán a lo largo de esta memoria serán creadas empleando este programa.

Llegados a este punto, ya tenemos un mapa correctamente entrenado. Ahora sólo falta obtener individuos producidos por un algoritmo evolutivo y proyectarlos sobre el mismo. Para llevar a cabo esta proyección utilizaremos el programa `umat`. En los siguientes capítulos, 4 y 5, se pueden encontrar variados ejemplos de proyección. Los individuos pueden proyectarse de uno en uno o varios de ellos a la vez, pudiendo de esta forma cumplir con los diversos objetivos que nos planteamos en la sección 3.1. Para diferenciar los individuos puede utilizarse cualquier tipo de cadena de caracteres como: un identificador único, el valor de fitness de un individuo o cualquier otra que nos pueda interesar.

Ahora que ya hemos creado un mapa y lo hemos probado proyectando individuos o poblaciones completas sobre él podemos crear un segundo mapa con la información obtenida a partir del primero. De esta forma podemos cumplir ciertos objetivos, tal como: centrarnos en ciertas áreas interesantes, descartar áreas que sean imposibles de visitar debido a algún tipo de restricción o modificar la forma que tiene el mapa de proyectar individuos para hacerlo más atractivo, visualmente hablando, modificando su función de densidad de probabilidad. La creación de un segundo mapa a partir de la información obtenida con un primero se ha utilizado en las secciones 4.1.1 y 4.1.2 para mejorar nuestra percepción del proceso evolutivo, y en la sección 4.2.1 para eliminar zonas poco interesantes o imposibles de visitar debido a restricciones del problema.

3.4. Conclusiones

Este capítulo se han descrito las posibilidades que la visualización puede aportar al campo de la computación evolutiva, y a la vez se han visto las limitaciones que lo afectan.

Para comenzar se ha hecho una clasificación de las técnicas de visualización en base a dos criterios:

- ¿Qué se ve? El genotipo, que es estructura de datos que representa una solución, o el fenotipo, que es la calidad de una solución.

- ¿Cuánto se ve? De cuántas generaciones proceden los datos a visualizar.

Debido a las limitaciones de la visión humana, en segundo lugar se han enumerado y descrito las técnicas de escalado multidimensional más importantes que nos permitirán reducir el número de dimensiones de un problema de cara a su visualización. De todas ellas el Mapas Autoorganizativos de Kohonen es la que mejor se adapta a nuestro objetivo por su buen funcionamiento con un elevado número de dimensiones, porque permiten proyectar cualquier punto desde el espacio de entrada a una representación bidimensional una vez entrenados, por estar disponible de forma gratuita en la red y por ser fácil de utilizar.

En último lugar se ha descrito el método de visualización multidimensional basado en el SOM propuesto en la presente memoria, cuyos funcionamiento a grandes rasgos sigue los estos pasos: elección del conjunto de entrenamiento, entrenamiento y verificación del SOM, y proyección de soluciones sobre el SOM.

Capítulo 4

Espacios de búsqueda

Una de las cosas que se puede llevar a cabo mediante el uso de un SOM es visualizar una representación del espacio de búsqueda. Según el problema con que tratemos y lo que más convenga a nuestros intereses existen dos posibilidades en cuanto a la forma de seleccionar el conjunto de entrenamiento:

1. Entrenar el mapa con un conjunto de puntos seleccionados del espacio de búsqueda del problema. De esta forma obtendremos una imagen de todo el espacio de búsqueda del problema. Según cómo de aleatoria sea esta selección de puntos y las distribuciones de probabilidad que utilicemos para escogerlos podemos obtener mapas más o menos fácilmente interpretables (Romero *et al.* , 2002).

2. Utilizar puntos obtenidos del algoritmo evolutivo durante la resolución del problema en cuestión. De esta segunda forma lo que obtendremos será una imagen de la parte del espacio de búsqueda explorado por el algoritmo evolutivo al tratar de resolver el problema. De está forma podemos incluso comparar diferentes ejecuciones del algoritmo para ver si siguen caminos parecidos o totalmente diferentes y así poder saber algo acerca de la sensibilidad del algoritmo al tamaño de la población y otros parámetros iniciales (Romero *et al.* , 2001).

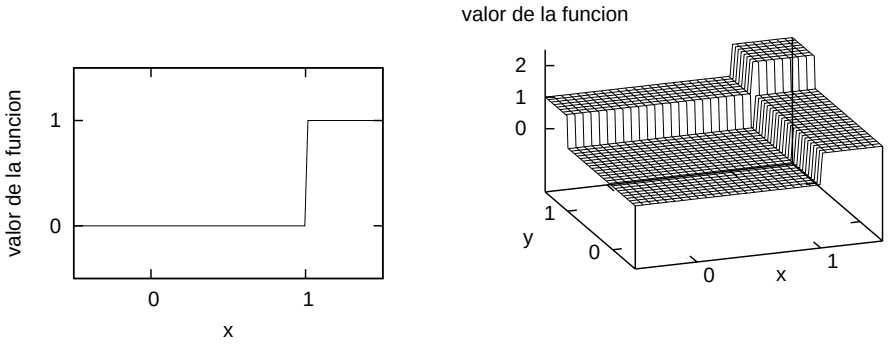
Estas dos posibilidades las estudiaremos debido a otra de las características beneficiosas del SOM: no sólo permite reducir en número de dimensiones

del espacio de entrada, sino que, además, es capaz de representar la función de densidad de probabilidad de dicho espacio (Kohonen, 1990), aunque no siempre de forma perfecta. De esta forma, además de poder proyectar sobre mapas entrenados con puntos seleccionados de forma aleatoria del espacios de búsqueda, podremos influir en la forma que dichos mapas toman mediante la selección cuidadosa de los conjuntos de entrenamiento para el SOM. Para ello aprovecharemos la información que tengamos a priori sobre un el espacio de búsqueda de un cierto problema o la información que podamos recolectar a través de los diversos intentos de resolverlo.

En las próximas secciones de este capítulo veremos algunos ejemplos de visualización de espacios de búsqueda empleando las dos posibilidades anteriormente descritas para varios problemas de prueba utilizados habitualmente en computación evolutiva tales como el problema onemax, la función de Rastrigin y el problema de la mochila.

4.1. Visualizando el espacio de búsqueda

Este es uno de los tipos de imagen que más y mejor información puede proporcionarnos acerca de un problema. Nos permitirá distinguir las mejores zonas del espacio de búsqueda de las peores sin más que observar en donde se proyectan los mejores individuos y los peores. Además podremos seguir la evolución del algoritmo a lo largo de su evolución y aprender sobre su comportamiento: la correcta creación de de la población inicial, el grado de diversidad de la población y su mantenimiento a lo largo de la evolución, la velocidad con que se produce el fenómeno de la convergencia (si es que se da) y hacia qué áreas, si el grado de presión selectiva es la adecuada, si el número de individuos es el adecuado, si los operadores genéticos cumple con las funciones para los que han sido diseñados y si las condiciones de terminación son las adecuadas.



(a) 1 variable (b) 2 variables

Figura 4.1: Gráficos de la función onemax para 1 y 2 variables.

4.1.1. El espacio de búsqueda del problema onemax

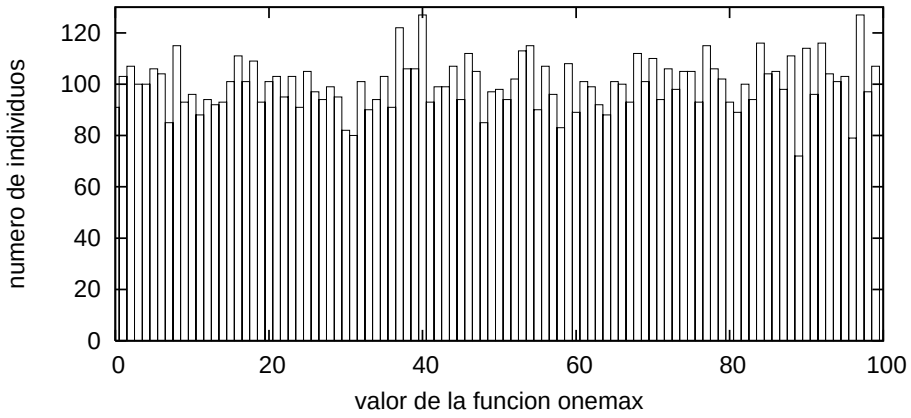
Dada una cadena binaria de longitud n, este problema consiste en maximizar la función

$$f(x) = \sum_{i=1}^{n} x_i \qquad x_i \in \{0,1\}^n \qquad x \in \mathbb{Z}_2^n$$

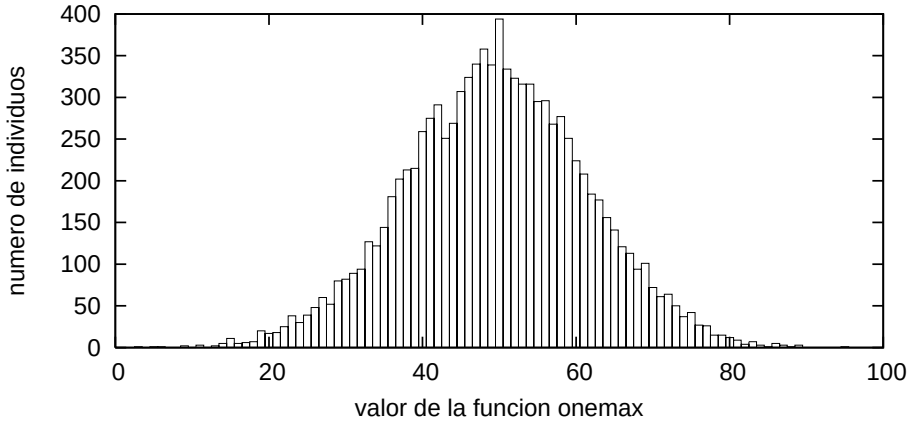
Esta es una función muy utilizada por los investigadores del campo de la computación evolutiva como problema de prueba (Garnier *et al.* , 1999; Harik *et al.* , 1999) y nos será útil para explicar la utilidad del SOM para visualizar espacios de búsqueda. La función que se busca maximizar es unimodal y relativamente fácil de resolver cuando el número de dimensiones es pequeño. A medida que el número de dimensiones crece, el número de óptimos locales crece exponencialmente con lo cual puede tardarse en localizar el óptimo global, especialmente en caso de utilizar únicamente operadores genéticos clásicos como el que cambia el valor de un bit aleatoriamente y el cruce monopunto. En la figura 4.1 podemos ver los gráficos para 1 y 2 variables de esta función.

El espacio de búsqueda de este problema está compuesto por todas las cadenas binarias de longitud n representadas por el espacio de los vectores binarios $\mathbb{Z}_2^n$. Los elementos de $\mathbb{Z}_2^n$ son vectores $z = (z_1, z_2, \ldots, z_n)$ con $z_i \in \{0,1\}$.

Al par de elementos de $\mathbb{Z}_2^n$ compuestos únicamente por ceros y unos suele denominárseles de forma abreviada **0** y **1**, respectivamente. Ellos representan la mejor (**1**) y peor (**0**) soluciones posibles al problema. En los siguientes experimentos se utilizará $n = 100$ para comprobar si es posible "aplanar" de alguna forma un espacio de dimensión 100 a 2 dimensiones.



(a) Distribución uniforme



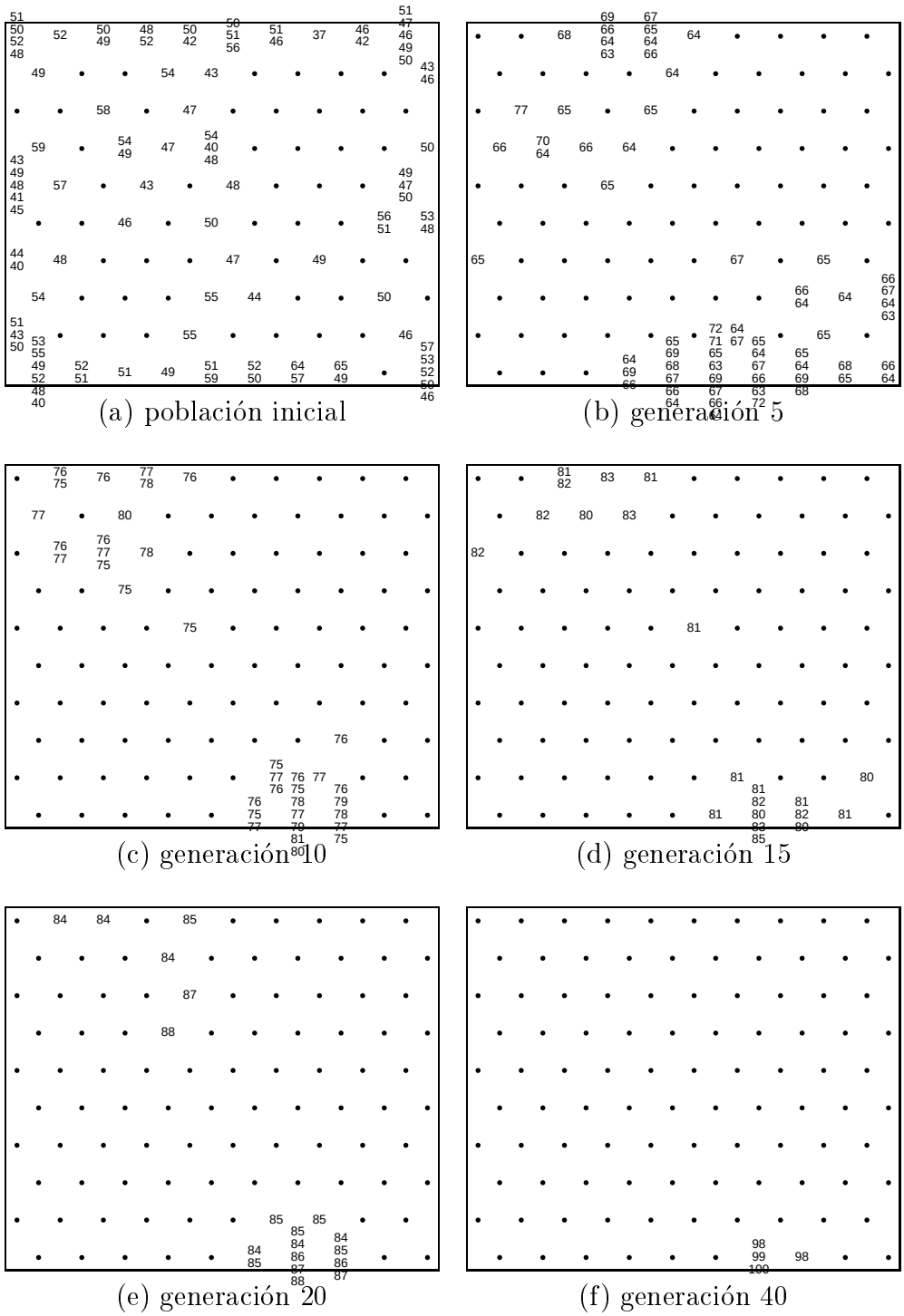
(b) Distribución normal

Figura 4.2: Comparativa de las distribuciones de dos conjuntos de entrenamiento para el problema onemax, cada uno de ellos con 10000 muestras. (a) Distribución uniforme (b) Distribución normal.

En primer lugar debemos entrenar el SOM y para ello necesitamos un conjunto de entrenamiento. La forma más inmediata para obtenerlo es se-

leccionar puntos de forma aleatoria del hipercubo $\mathbb{Z}_2^{100}$ siguiendo una distribución uniforme. En la figura 4.2a se puede ver un histograma con la distribución de valores de la función onemax de los 10000 puntos seleccionados para entrenar el SOM. Una vez entrenado el mapa siguiendo los pasos que se describen en la sección 3.3, en nuestro caso utilizando el **SOM_PAK** (Kohonen *et al.* , 1996), se proyectarán las soluciones obtenidas en cada generación del algoritmo genético simple programado para resolver el problema mediante el SOM entrenado con un conjunto de entrenamiento de puntos seleccionados aleatoriamente. Como podemos ver en la figura 4.3 este método no proporciona buenos resultados. La población inicial se escoge de forma adecuada ya que ocupa el 50 % las neuronas del mapa. Como era de esperar sus valores iniciales están en torno a 50 pues la probabilidad inicial de que cada bit sea 0 ó 1 es del 50 %. También nos muestra cómo sucede el fenómeno de la convergencia de la población. En contra de lo esperado, el proceso no hace que la población se concentre en una unica área, sino que hasta casi el final del proceso, en la generación 40, está dividida en dos zonas de similar valor de fitness. Aunque sí es cierto que estas dos zonas son cada vez más reducidas, y que al final, la mejor de ellas hace converjer hacia sí a toda la población. El principal problema con este mapa es que proyecta individuos que representan soluciones muy diferentes, tanto como 50 y 100, en la misma neurona.

Para mejorar las características visuales del SOM la metodología que aquí proponemos se basa en escoger el conjunto de entrenamiento de forma que se altere la función de densidad de probabilidad del espacio de entrada de una forma más conveniente que la ya vista. Ahora, en lugar de escoger puntos de $\mathbb{Z}_2^{100}$ de forma totalmente aleatoria, vamos a tener en cuenta qué solución representa cada punto dentro de dicho hipercubo. Dentro de $\mathbb{Z}_2^{100}$ hay exactamente $\binom{100}{x}$ puntos cuyo valor de la función onemax es x. Así, el número de puntos con un cierto valor de la función onemax crece a medida que nos alejamos de los puntos **0** y **1**, de los cuales sólo hay 1, y se hace máximo al acercarnos al valor 50, del cual hay $\binom{100}{50}$. Teniendo en cuenta estas probabilidades se escogió un nuevo subconjunto aleatorio de $\mathbb{Z}_2^{100}$ siguiendo una



(a) población inicial (b) generación 5

(c) generación 10 (d) generación 15

(e) generación 20 (f) generación 40

Figura 4.3: Proyección de onemax sobre un SOM entrenado con puntos aleatorios de $\mathbb{Z}_2^{100}$ que siguen una distribución uniforme. Los números dentro del gráfico indica el valor de la función onemax de cada individuo.

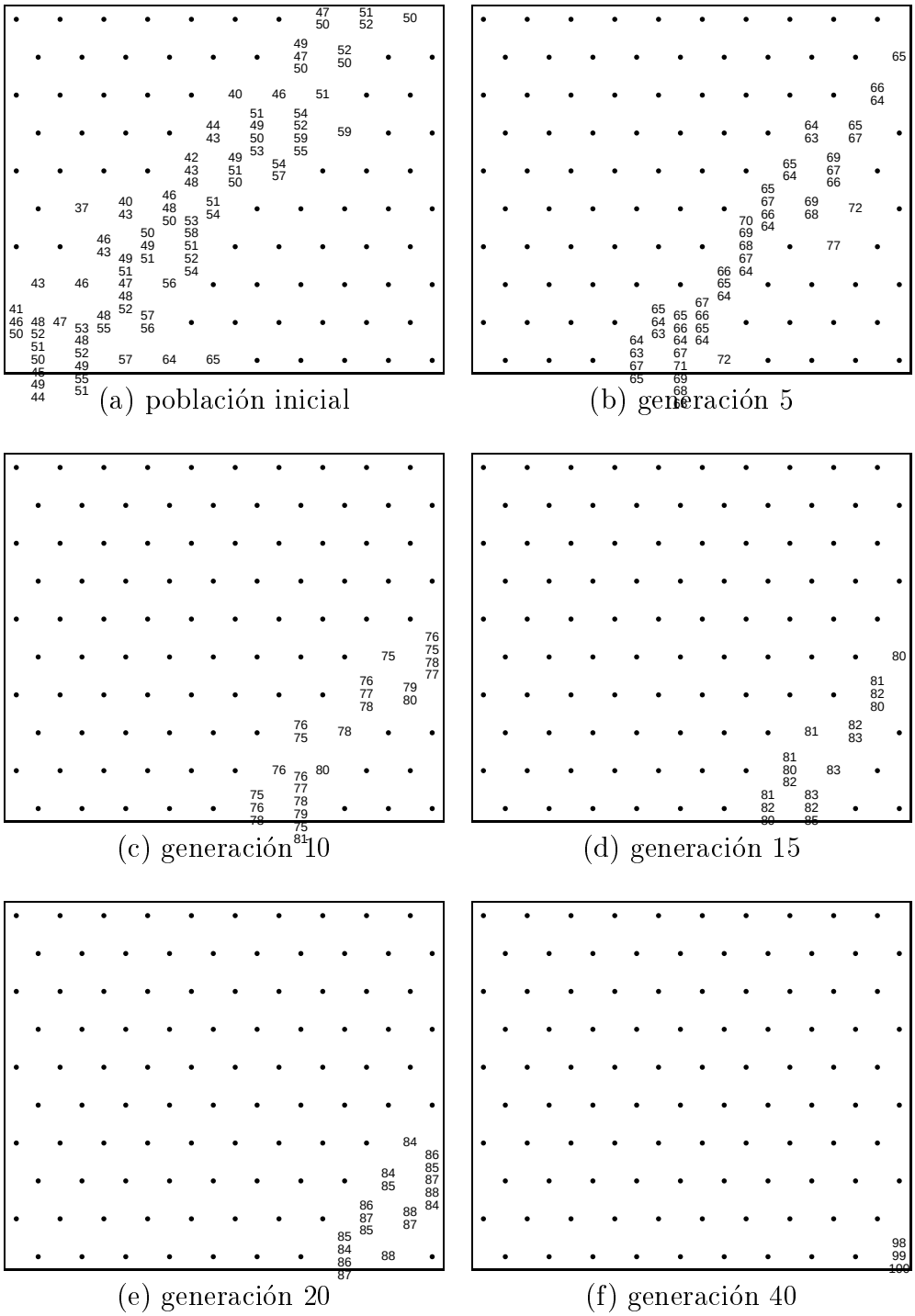
distribución normal. En la figura 4.2b podemos encontrar un histograma con la distribución de valores de fitness de este conjunto de datos, y además, podemos compararlo con la distribución utilizada anteriormente. El resultado de la proyección del mismo algoritmo genético visto en la 4.3 puede verse ahora proyectado sobre el nuevo SOM en la figura 4.4.

Estudiando esta nueva secuencia se observa que el punto **0**, el peor, queda situado sobre la esquina superior-izquierda y el **1**, el mejor, en la inferior-derecha, y que el resto de valores se distribuyen de forma continua entre ambas esquinas. Ahora la población inicial en vez de ocupar la mayor parte del mapa,como en la población inicial de la figura 4.3, ocupa solamente la franja comprendida entre las esquinas inferior-izquierda y la superior-derecha, es decir, el área con valores de la función próximos a 50, ver la población inicial de la figura 4.4. Esto ocurre por el mismo motivo explicado antes: la inicialización aleatoria de los individuos. La evolución de la población ahora sí toma una clara dirección hacia el óptimo, cuyo valor es 100, y al cual se acerca cada vez más a medida que avanza la evolución. Ahora el proceso de convergencia toma una dirección clara y bien definida hacia el óptimo, cosa que no sucedía antes. Tampoco ocurre que haya varias zonas de fitness parecido que estén separadas. Este fenómeno es visible lo largo las generaciones en la figuras 4.3 de forma que hay dos zonas con soluciones con valores de fitness similares tanto arriba como abajo, cosa que no ocurre en la figura 4.4.

Para este problema ha sido posible encontrar una forma de visualización satisfactoria gracias al conocimiento a priori que teníamos de la forma del espacio de búsqueda ya que el primer intento no produjo resultados satisfactorios.

4.1.2. El espacio de búsqueda de la función de Rastrigin

Esta es otra función muy empleada como función de prueba en el campo de la computación evolutiva. En la ecuación, 4.1, suele variarse el número de dimensiones según la dificultad que se le quiera dar al problema, creciendo

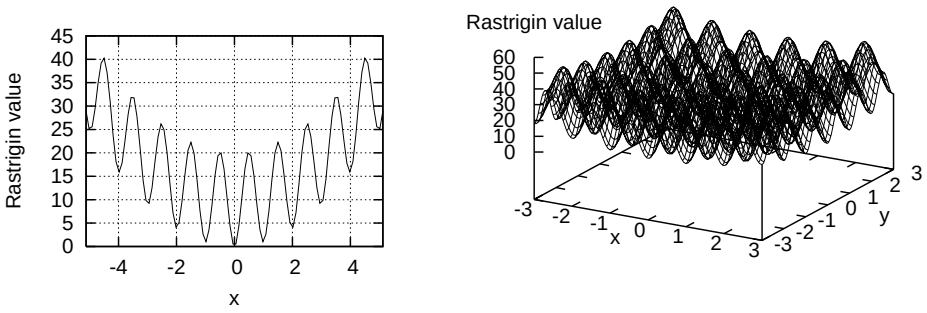


(a) población inicial (b) generación 5

(c) generación 10 (d) generación 15

(e) generación 20 (f) generación 40

Figura 4.4: Proyección de onemax sobre un SOM entrenado con puntos aleatorios de $\mathbb{Z}_2^{100}$ que siguen una distribución normal centrada en 50 basada en el número de individuos con un cierto valor de fitness. Los números dentro del gráfico indica el valor de la función onemax de cada individuo.



(a) 1 variable (b) 2 variables

Figura 4.5: Función de Rastrigin con 1 y 2 variables.

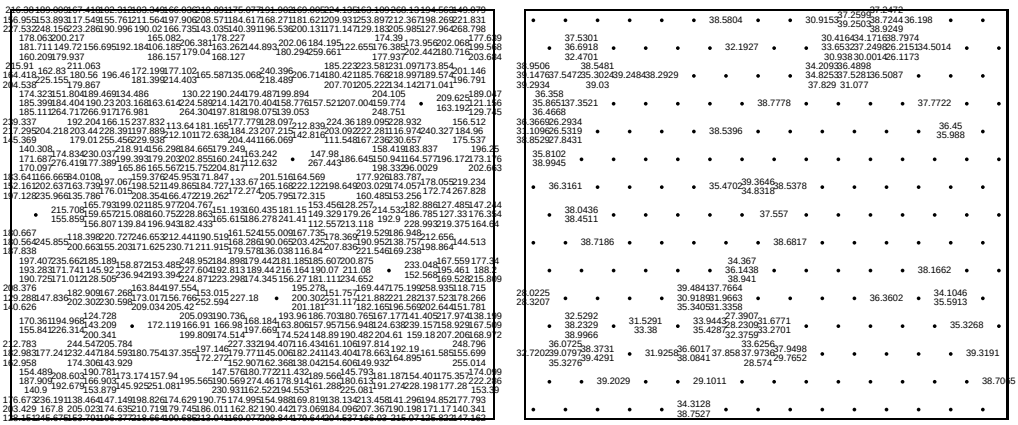
ésta de forma exponencial con el número de dimensiones. α es una constante para controlar la altura de la función cuyo valor más habitual es 10 y n es el número de dimensiones. En los siguientes experimentos con esta función emplearemos los valores $\alpha = 10$ y $n = 10$.

$$f(x) = \alpha n + \sum_{i=1}^{n} x_i^2 - \alpha \cos(2\pi x_i) \qquad x_i \in [-5{,}12, 5{,}12] \tag{4.1}$$

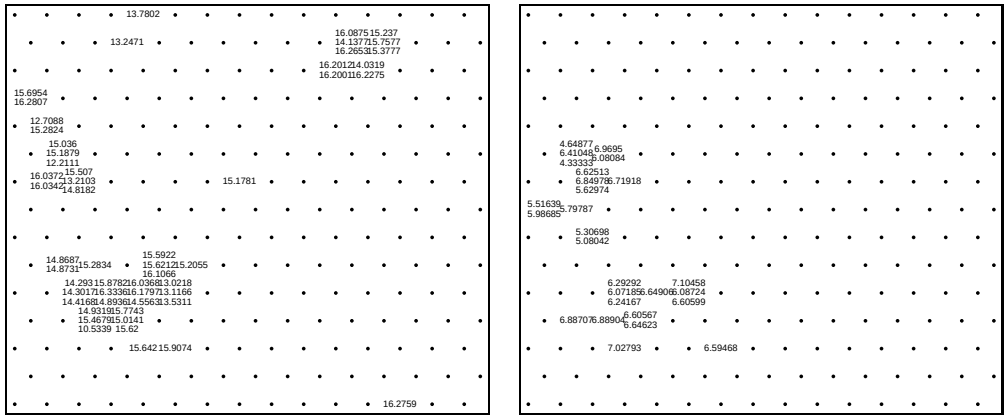
Esta función se caracteriza por la existencia de multitud de subóptimos cuyos valores van creciendo a medida que nos alejamos del mínimo global situado en la el punto $\mathbf{0}$. Podemos ver su forma en 1 y 2 dimensiones en la figura 4.5.

Al igual que hicimos antes con la función onemax, empezaremos entrenando el SOM con un conjunto de puntos escogidos aleatoriamente del espacio de búsqueda, cuya forma es ahora una hiperesfera de centro $\mathbf{0}$ y radio 5,12 de $\mathbb{R}^{10}$. El resultado de proyectar los individuos generados durante la ejecución de un algoritmo evolutivo para tratar de buscar el mínimo global de la función puede verse en la figura 4.6.

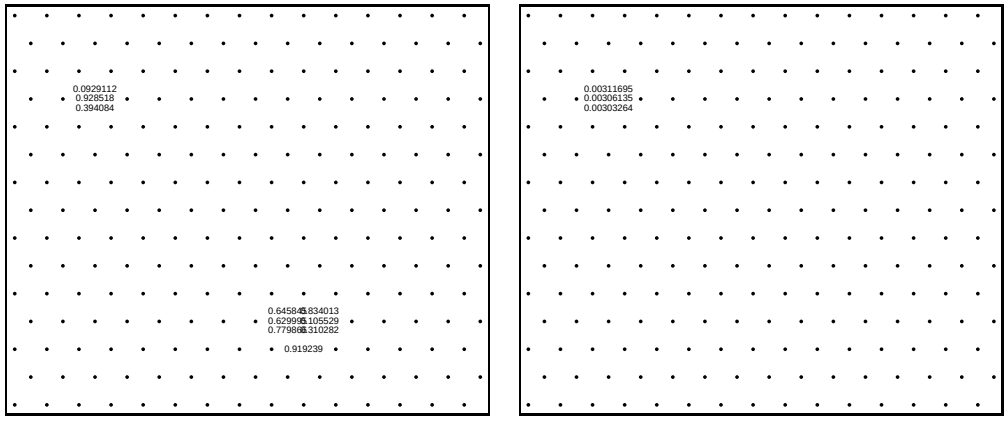
De la figura 4.6 podemos extraer información interesante: La población inicial se escoge de forma adecuada ya que ocupa el 97.3 % de la representación de todo el espacio de búsqueda. De nuevo, al igual que con onemax, se puede observar el fenómeno de convergencia hacia la solución y con



(a) población inicial (b) generación 10



(c) generación 20 (d) generación 30



(e) generación 50 (f) generación 100

Figura 4.6: Proyección de individuos de un algoritmo evolutivo que busca el mínimo de la función de Rastrigin sobre un SOM entrenado con puntos aleatoriamente escogidos de $\mathbb{R}^{10}$ siguiendo una distribución uniforme. Los números dentro de los gráficos indican el valor de la función de cada individuo.

el mismo defecto: individuos casi iguales caen en zonas alejadas del SOM, como puede verse en las generaciones 20, 30 y 50. Como era previsible, el área cubierta por la población va decreciendo a medida que el algoritmo se acerca a la solución óptima. Como defecto, observar que en esta proyección individuos tanto de la población inicial como de la final caen dentro de la misma neurona del SOM, ya pesar de tener aptitudes muy diferentes pueden tener representaciones muy parecidas.

Para tratar de mejorar las características visuales del SOM lo entrenaremos de nuevo teniendo en cuenta no sólo el espacio de búsqueda sino la información adicional que nos proporciona la función de Rastrigin. Tras las primeras generaciones es fácil comprobar que la población se acerca al centro de la hiperesfera de centro 0 y radio 5.12 de $\mathbb{R}^{10}$ a la vez que abandona su corteza. Así pues, en este segundo intento al entrenar el SOM lo haremos escogiendo puntos aleatorios de $\mathbb{R}^{10}$ siguiendo una distribución normal centrada en 0, es decir, con mayor probabilidad cerca del centro que del límite exterior de la hiperesfera.

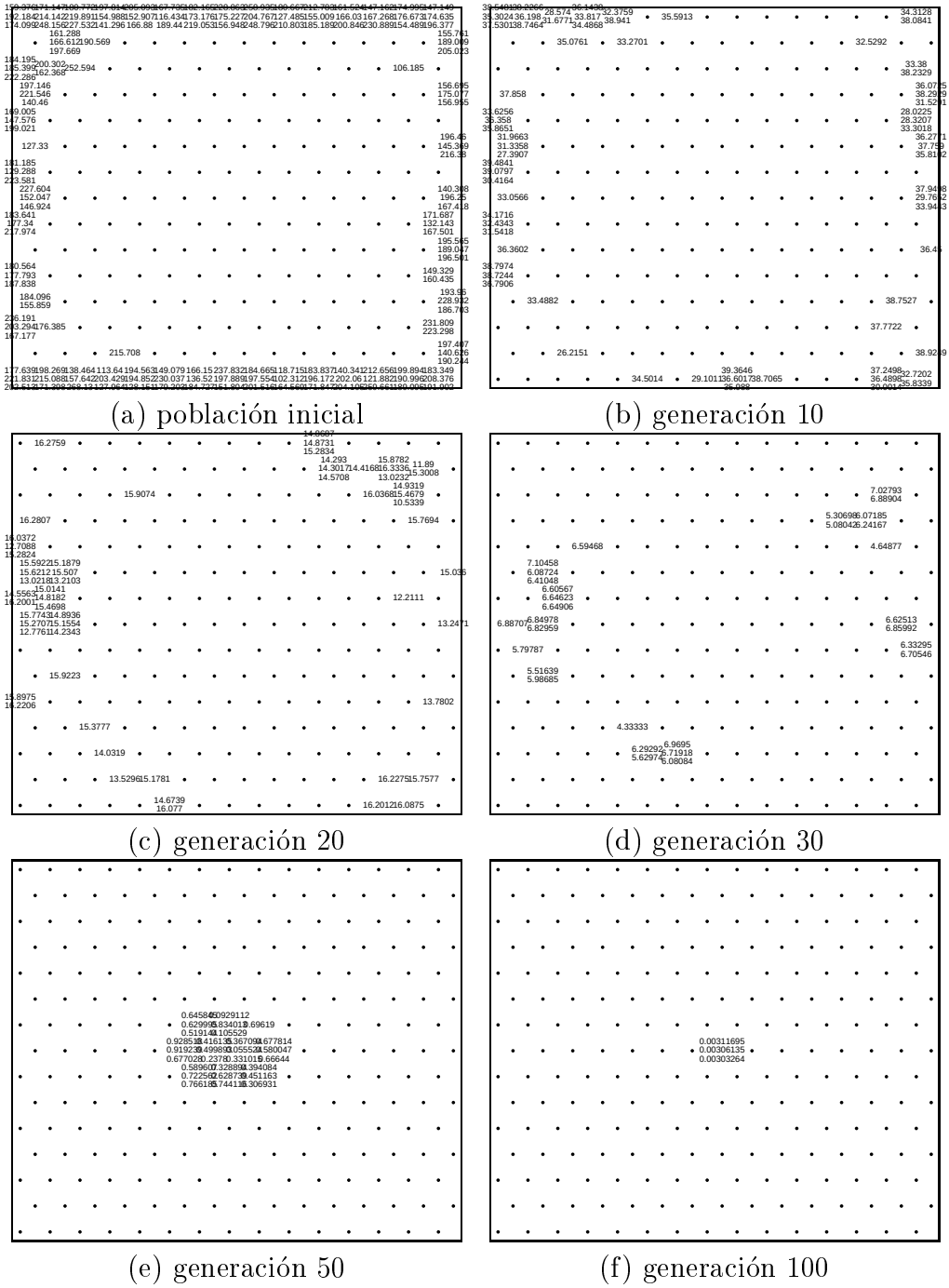
El resultado de esta segunda proyección podemos verlo en la figura 4.7. El proceso de convergencia, a diferencia que antes, toma una clara dirección desde el borde hacia el centro del mapa. Así pues, la población inicial ocupa el borde del mapa y se dirige formando círculos casi concéntricos hacia el centro a la vez que se acerca al óptimo global generación tras generación.

Esta segunda forma de proyección es más fácil de comprender y permite el seguimiento de la población de una forma más intuitiva y natural:

- La forma de mostrar la población inicial es muy diferente en ambos casos: repartida por todo el mapa en el primer caso, figura 4.6(0), y alejada del centro en el segundo, figura 4.7(0).

- El proceso de convergencia es difícil de seguir con el primer mapa ya que no toma una dirección claramente definida. El segundo mapa por contra lo hace extremadamente sencillo pues toma un clara dirección hacia el centro.

- En el primer mapa, individuos con muy diferentes valores de fitness



(a) población inicial (b) generación 10

(c) generación 20 (d) generación 30

(e) generación 50 (f) generación 100

Figura 4.7: Proyección de individuos de un algoritmo evolutivo que busca el mínimo de la función de Rastrigin sobre un SOM entrenado con puntos aleatoriamente escogidos de $\mathbb{R}^{10}$ siguiendo una distribución normal centrada en 0. Los números dentro del gráfico indican el fitness del individuo. Los números del interior de los gráficos indican el valor de la función de cada individuo.

son proyectados en zonas cercanas, y, por contra, otros muy parecidos son repartidos sin orden aparente por regiones muy separadas. En el segundo mapa se aprecia una fuerte estructuración concéntrica de los valores de los individuos, su valor de fitness aumenta a medida que se acercan al centro.

4.2. Visualizando el espacio explorado durante la búsqueda

Hasta ahora hemos visto representaciones del espacio de búsqueda completo de ciertos problemas. En el siguiente experimento lo que haremos será representar sólo la porción de dicho espacio explorada por un algoritmo evolutivo durante la búsqueda de la solución a un problema.

Para ello lo que haremos será utilizar un conjunto de entrenamiento diferente para entrenar los mapas de Kohonen de los utilizados hasta ahora. En la sección anterior, 4.1, hemos utilizado conjuntos de entrenamiento formados mediantes puntos seleccionados de forma aleatoria de todo el espacio de búsqueda de un problema con una cierta distribución de probabilidad. Ahora, en cambio, los puntos los seleccionaremos de la propia ejecución de un algoritmo evolutivo que resuelve un problema. Así en vez de representar todo un espacio de búsqueda seremos capaces de centrarnos sobre las áreas exploradas por nuestros algoritmos.

4.2.1. El espacio explorado en el problema de la mochila

En el problema de la mochila binaria un conjunto de n objetos se deben introducir en una mochila de capacidad C. Cada objeto tiene un peso P_i y un beneficio B_i asociados, y el problema consiste en seleccionar un subconjunto de objetos cuyo peso total no exceda la capacidad de la mochila y cuyo beneficio sea máximo.

Hay una gran variedad de problemas de este tipo, muchos de los cuales son conocidos por ser NP-duros. Para nuestros experimentos hemos escogido un problema de la mochila binaria descrito en (Michalewicz, 1996) página. 81, en su variedad fuertemente correlacionada, con capacidad media y empleando un algoritmo de penalización lineal para calcular el fitness y asegurar la validez de los resultados. Existen multitud de métodos para resolver este problema, en (Martello *et al.* , 1997) podemos ver una recopilación de los métodos algorítmicos más importantes, y en (Hinterding, 1994) se presentan varias formas de resolución mediante algoritmos evolutivos. A continuación describiremos brevemente el este problema:

Dados un conjunto de pesos P, otro de beneficios B y una capacidad C, el problema consiste en encontrar un vector binario $x \in \mathbb{Z}_2^n$ tal que

$$\sum_{i=1}^{n} x_i \cdot P_i \leq C$$

y para el cual la suma de beneficios

$$\sum_{i=1}^{n} x_i \cdot B_i \tag{4.2}$$

sea máxima.

El conjunto de pesos, P, se ha escogido de forma aleatoria entre 1 y 10 y el de beneficios, B, se calcula a través de P mediante la fórmula

$$B_i = P_i + 5 \tag{4.3}$$

La función de penalización lineal empleada ha sido:

$$pen(x) = \rho \cdot \left(\sum_{i=1}^{n} x_i \cdot P_i - C \right) \qquad \rho = max_{i=1..n}\{B_i/P_i\} \tag{4.4}$$

En los siguientes experimentos con este problema las soluciones se representarán mediante vectores binarios de longitud $n = 500$, con lo cual el espacio de búsqueda será el hipercubo $\mathbb{Z}_2^{500}$, y utilizaremos una capacidad de

mochila de $C = 1389$ y un valor de $\rho = 5$. En la sección A.2 mostramos el conjunto de pesos utilizado.

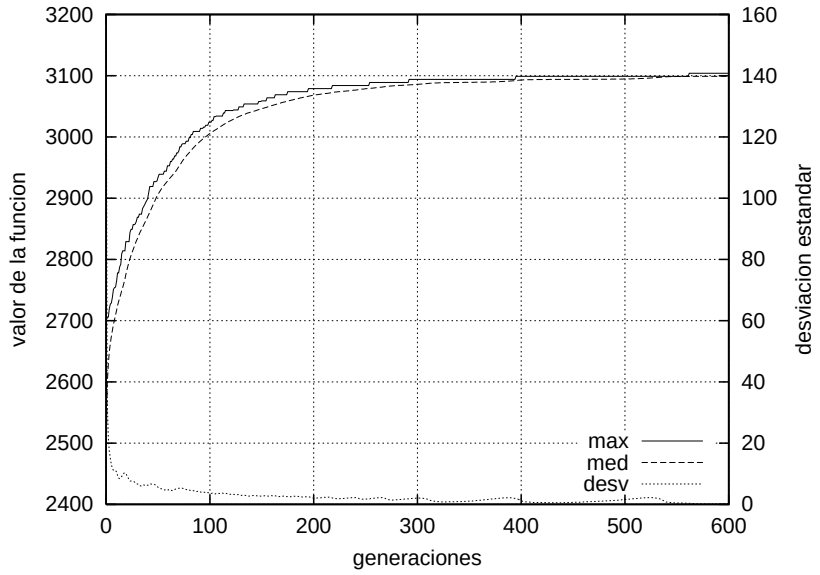
En esta instancia del problema, el valor del óptimo global es 3109, y a pesar de haber $\binom{55}{42} \cdot \binom{45}{2} \approx 1{,}44 \cdot 10^{15}$ diferentes combinaciones de objetos en la mochila que nos permiten conseguirlo, a duras penas hemos conseguido obtenerlo en unas pocas ejecuciones de nuestro algoritmo. El máximo valor que se obtiene habitualmente es 3104, que es un subóptimo mucho más abundante que el óptimo global. Esto se debe sin duda a la dificultad del problema, ante cuyo espacio de búsqueda de casi 2^{500} posibles combinaciones (algo menor en realidad debido a las restricciones), el número de óptimos es insignificante.

El primer paso para visualizar el espacio de búsqueda será crear un SOM para proyectar los individuos generados por el algoritmo evolutivo escrito para resolver este problema. Hasta el momento hemos entrenado los mapas utilizado conjuntos de entrenamientos cuyos puntos se seleccionaban aleatoriamente del espacio de búsqueda. Ahora en cambio vamos a probar a entrenarlos con individuos obtenidos durante la resolución del problema. De esta forma, en vez de obtener una representación de todo el espacio de búsqueda, obtendremos una representación del espacio explorado por el algoritmo evolutivo en su búsqueda de la solución. Esta forma puede suponer un ventaja, especialmente en problemas como este, cuyo espacio de búsqueda es muy extenso.

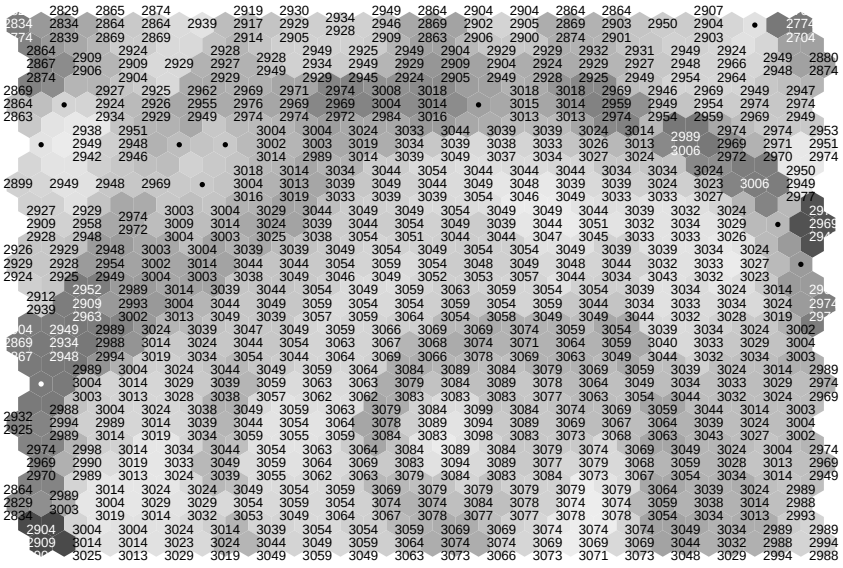
Para obtener el conjunto de entrenamiento se han ejecutado el algoritmo 100 veces con diferentes parámetros iniciales, escogiendo 10000 individuos de forma aleatoria de dichas ejecuciones. De esta forma se trata de amoldar la densidad de probabilidad del espacio del espacio de búsqueda a las regiones visitadas por el algoritmo durante su funcionamiento. Los resultados producidos mediante esta forma de entrenar un SOM podemos verlos en la figura 4.9.

De las figuras 4.8 y 4.9 podemos extraer varias conclusiones acerca del funcionamiento de nuestro algoritmo:

- El método de inicialización de la población utilizado no es el mejor

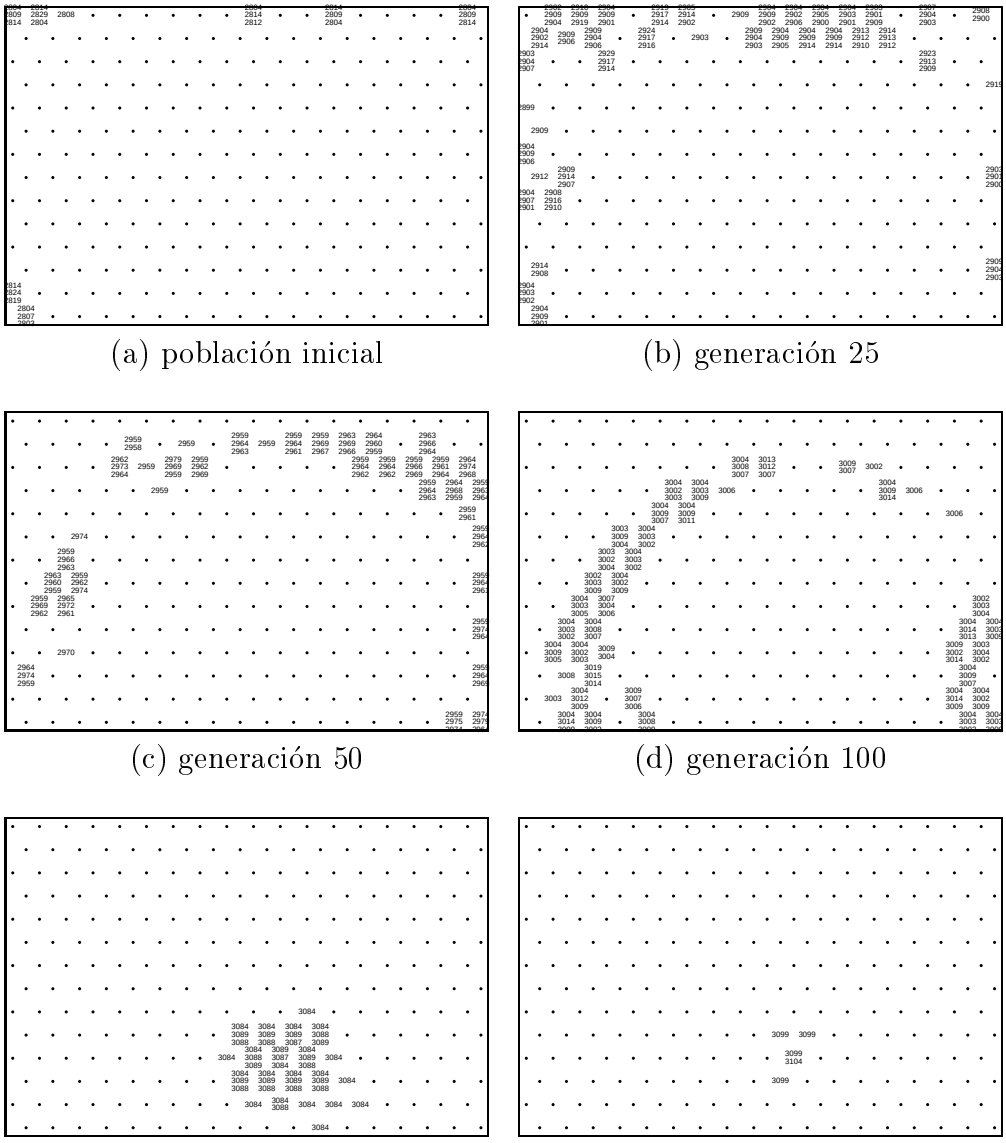


(a) Estadísticas de la ejecución



(b) SOM

Figura 4.8: SOM para proyección del problema de la mochila entrenado con 10000 individuos seleccionados al azar de 100 experimentos, representando el espacio explorado por el algoritmo durante su ejecución. (a) Estadísticas de la ejecución que puede verse en la figura 4.9. (b) SOM con el que se ha creado la proyección de la figura 4.9, los números indican el valor de fitness de los individuos proyectados en cada neurona.



(a) población inicial (b) generación 25

(c) generación 50 (d) generación 100

(e) generación 300 (f) generación 600

Figura 4.9: Proyección del problema de la mochila sobre el SOM entrenado con 10000 individuos seleccionados al azar de 100 experimentos de la figura 4.8, representando el espacio explorado por el algoritmo durante su ejecución.

posible. Se puede llegar a esta conclusión sin más que observar la distribución de la población inicial en la figura 4.9(a), pues sólo cubre el 3.17 % de toda la superficie que representa el espacio explorado durante la búsqueda. Dado que estamos empleando una variante del problema de la mochila de capacidad media, sería de esperar que las soluciones tuviese en media el 50 % de los objetos posibles. Esto no es así debido a la forma de la función de fitness, vease la ecuación 4.2, que no depende en exclusiva del peso, sino del beneficio asociado a cada peso definido por la ecuación 4.3. Según dichas ecuaciones, el beneficio asociado a introducir varios objetos ligeros es mayor que el de un único objeto de peso equivalente. De acuerdo con esto, en vez de hacer que cada bit del vector que representa las soluciones se inicialize con un 50 % de probabilidades de ser 0 ó 1, habría que incrementar la probabilidad de ser 1. Posteriores experimentos demostraron que el valor óptimo de inicialización a 1 era del 68.8 %, dado que las soluciones óptimas contienen 344 bits a 1 de los 500 posibles.

- El proceso de convergencia de la población es fácilmente observable y se produce desde los extremos hacia la zona inferior-central del mapa. De esta forma las buenas y malas zonas del espacio de búsqueda quedan bien delimitadas casi en forma de anillos concéntricos en la proyección creada mediante el programa `umat` en la que hemos mantenido el color del fondo que representa las distancias entre las neuronas del SOM, ver figura 4.8(b).

Como hemos visto esta forma de visualizar el espacio explorado por un algoritmo evolutivo nos permite seguir su funcionamiento y descubrir algunas de sus características más importantes. A continuación, para el problema de la mochila, intentaremos ver todo el espacio de búsqueda de nuestro problema, y no sólo la parte explorada por el algoritmo implementados por nosotros para resolverlo.

4.2.2. El espacio de búsqueda en el problema de la mochila

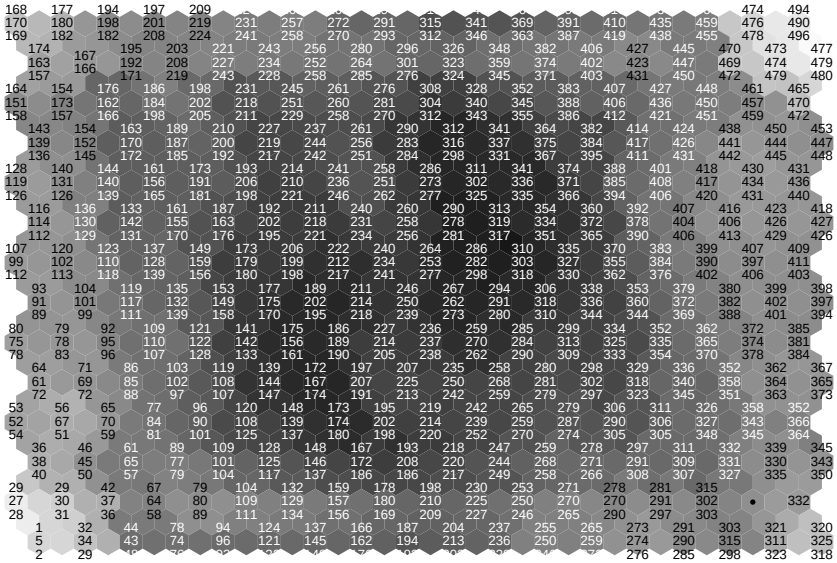
Para poder comparar la diferencia entre visualizar el espacio de búsqueda y el espacio explorado durante la búsqueda vamos a continuación a proyectar la misma ejecución que podemos ver en la figura 4.9 sobre un SOM entrenado según se ha visto en la sección 4.1, es decir, con puntos aleatorios del espacio de búsqueda. Para ello lo que hemos hecho ha sido seleccionar 10000 puntos de forma aleatoria de $\mathbb{Z}_2^{500}$ siguiendo una distribución uniforme. En la figura 4.11 podemos ver el resultado de la proyección sobre un SOM entrenado con este tipo de conjunto de datos.

En la figura 4.10(b) tenemos un mapa que proyecta todo el espacio de búsqueda $\mathbb{Z}_2^{500}$. Lo hace de forma que los individuos con menos unos caerán en la zona inferior-izquierda, y los que tienen más unos en la superior-derecha, distribuyendo los demás de forma diagonal entre ambas zonas. Como se ve en la figura 4.10(a), el mapa representa perfectamente todas las posibles combinaciones de ceros y unos y las distribuye de forma razonable entre las dos esquinas antes mencionadas, pero al fijarnos en la proyección de la población sobre el mismo, figura 4.10(b), vemos que no se ajusta a nuestros propósitos por varios motivos:

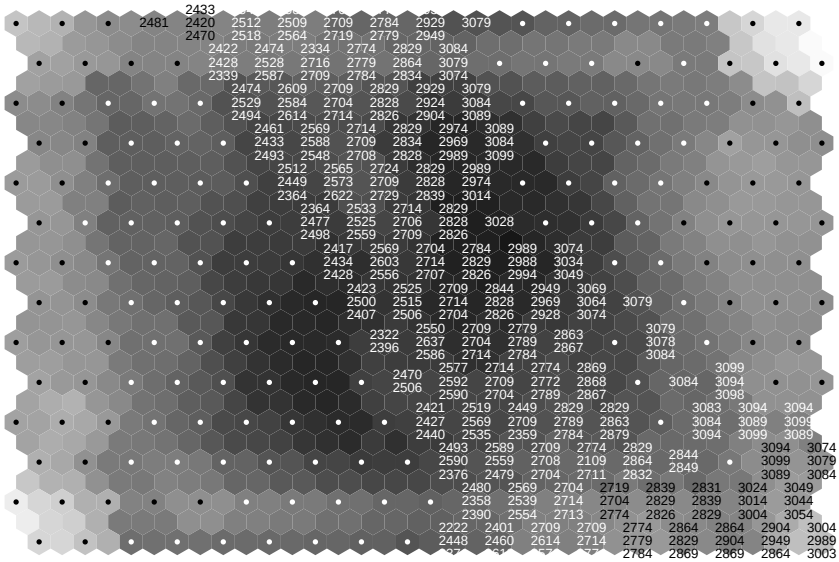
- El 44.4 % de área del mapa no es pobladas por ningun individuo, nos referimos a las esquinas inferior-izquierda y a la superior-derecha, con lo cual se produce un desperdicio de espacio, cuyas neuronas podría haberse utilizado para representar mejor las zonas interesantes del espacio de búsqueda.

- El área correspondiente a la esquina inferior-izquierda debería poblarse con los dividuos de muy bajo o nulo peso, que normalmente no son encontrados por el algoritmo. Esta es una de las zonas candidatas a desaparecer por no ser utilizada de forma que puede aprovecharse para ser poblada con otros individuos que si sean encontrados por el algoritmo.

- La esquina superior-derecha correspondería a los individuos de peso

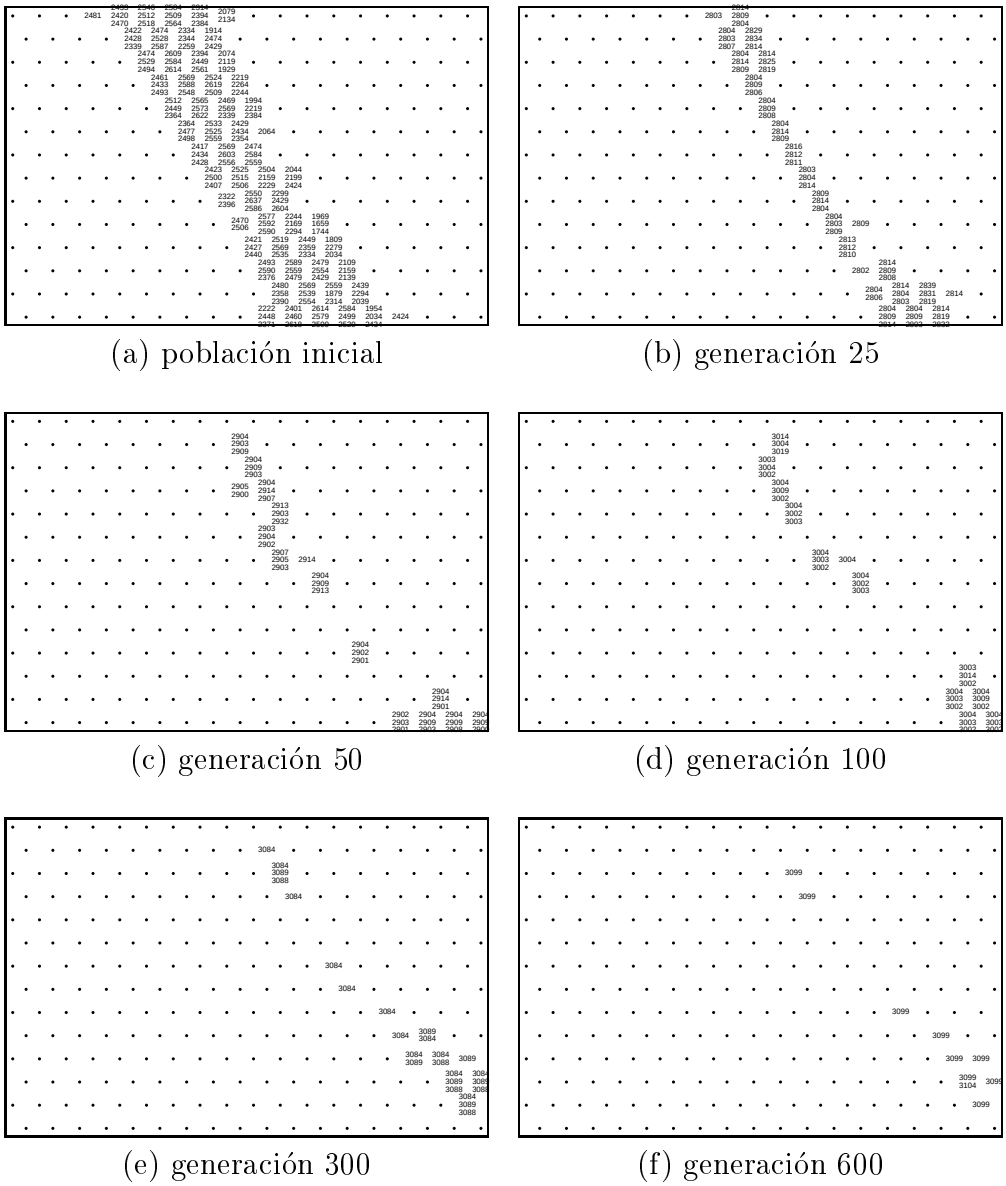


(a) Número de unos



(b) SOM

Figura 4.10: SOM para proyección del problema de la mochila entrenado con 10000 puntos seleccionados al azar de su espacio de búsqueda, $\mathbb{Z}_2^{500}$, siguiendo una distribución uniforme. Se puede ver la proyección en la figura 4.11. Los números en (a) indican el número de unos de los patrones de entrenamiento que caen sobre cada neurona, y en (b), el fitness de los individuos proyectados sobre cada neurona.



(a) población inicial (b) generación 25

(c) generación 50 (d) generación 100

(e) generación 300 (f) generación 600

Figura 4.11: Proyección del problema de la mochila sobre el SOM entrenado con puntos seleccionados al azar de su espacio de búsqueda, $\mathbb{Z}_2^{500}$, siguiendo una distribución uniforme de la figura 4.10b. Los números del interior de los gráficos a-f indican el valor de fitness.

muy elevado, que es imposible de alcanzar debido a la restricción de capacidad del problema, en concreto todas aquellas con más de 344 unos, además de ciertas combinaciones con un número menor de unos. Este problema se produce por que la representación de las soluciones, un vector binario de longitud 500, permite la creación de soluciones no válidas. Estas se van eliminando a medida que evoluciona la población debido al uso de una función de evaluación que penaliza a los individuos que superan la capacidad establecida (ecuación 4.4).

- El proceso de convergencia es difícil de apreciar debido a lo cerca que están las zonas con buenos y malos individuos. Seria mucho mejor si dicha distancia fuese la mayor posible dentro del mapa, es decir, si realmente se utilizasen todas las neuronas del SOM.

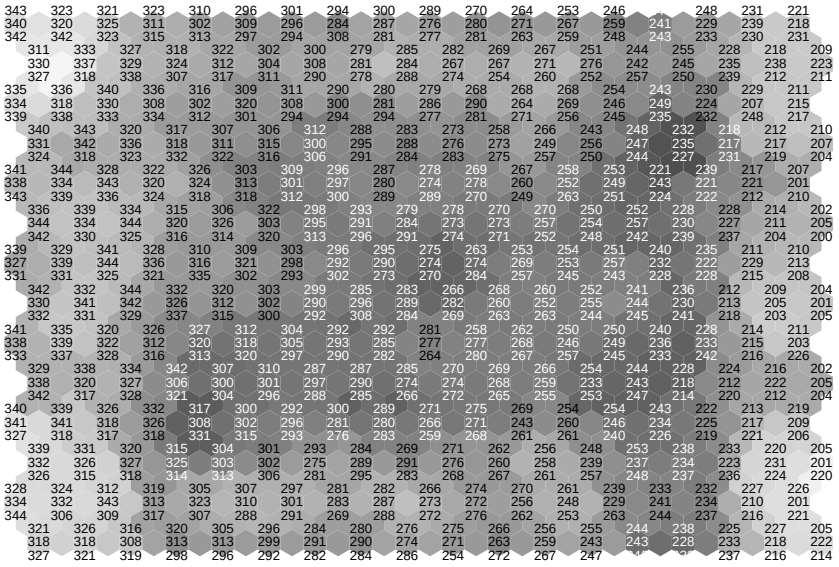
Partiendo de la información recabada en el primer intento de visualizar el espacio de búsqueda obtenida de las figuras 4.10 y 4.11, vamos a entrenar un nuevo SOM con un conjunto de datos diferente. Para ello definiremos el subconjunto $\mathbb{Z}'$, del que escogeremos los puntos, de la siguiente forma:

$$\mathbb{Z}' \subset \mathbb{Z}_2^{500}, \qquad \mathbb{Z}' = \left\{ x \in \mathbb{Z}_2^{500} \,\middle|\, \sum_{i=1}^{500} x_i > 200 \wedge \sum_{i=1}^{500} x_i < 345 \right\}$$

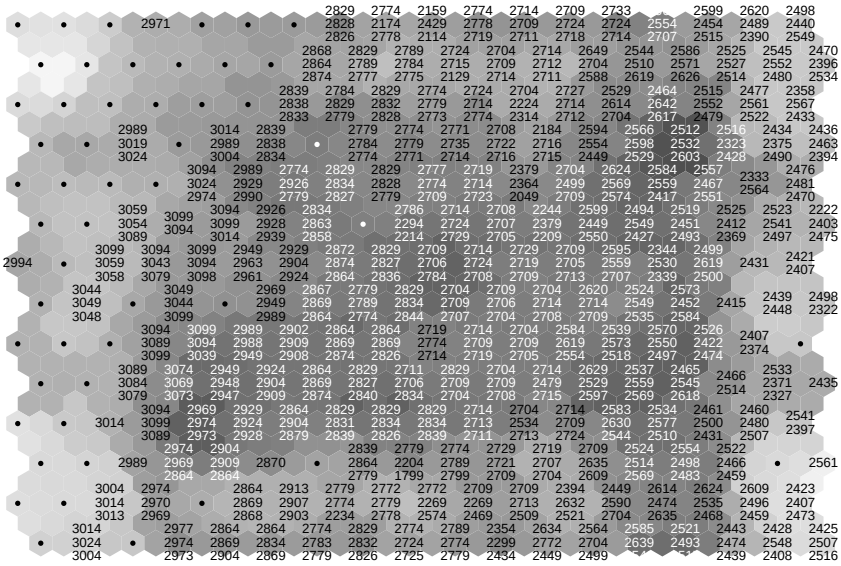
Al igual que antes, escogeremos 10000 valores al azar de $\mathbb{Z}'$ siguiendo una distribución uniforme y entrenaremos otro SOM con dicho conjunto. El resultado de proyectar sobre el mapa así entrenado la misma ejecución de nuestro algoritmo que en las figuras 4.9 y 4.11, lo podemos ver en la figura 4.13.

Comparando las figuras 4.11 y 4.13 podremos apreciar una mejora evidente en la calidad de la proyección y en la calidad de la información que proporcionan. En esta nueva proyección, se han descartado las zonas que no podía ser ocupadas por ser poco óptimas y también aquellas otras que incumplían la restricción de capacidad de la mochila. He aquí algunas de las conclusiones que podemos sacar de esta última proyección:

- La población evoluciona de derecha a izquierda, pasando de las áreas

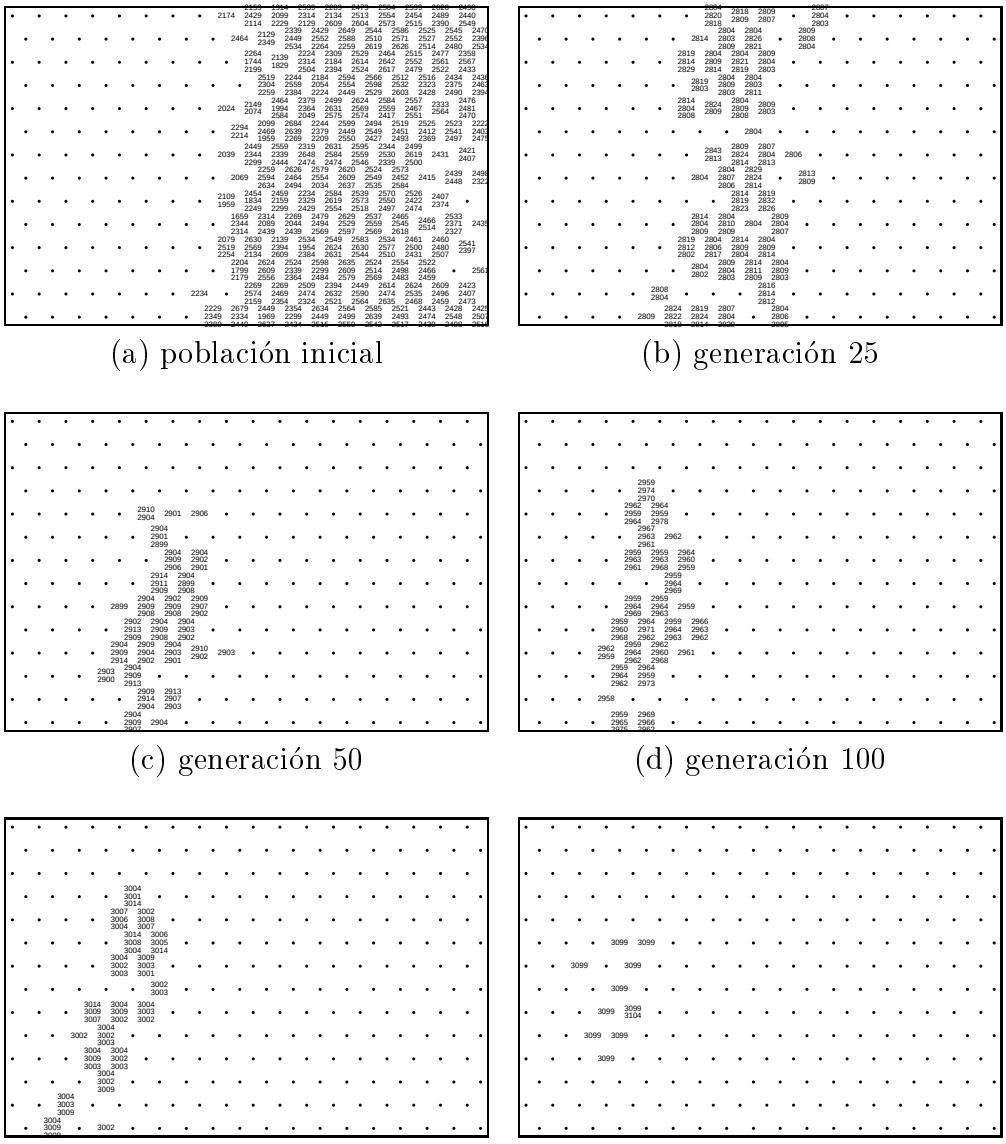


(a) Número de unos



(b) SOM

Figura 4.12: SOM para proyección del problema de la mochila entrenado con puntos seleccionados al azar del subespacio de búsqueda, $\mathbb{Z}'$, siguiendo una distribución uniforme. Los números en (a) indican el número de unos de los patrones de entrenamiento que caen sobre cada neurona, y en (b), el fitness de los individuos proyectados sobre cada neurona.



(a) población inicial (b) generación 25

(c) generación 50 (d) generación 100

(e) generación 300 (f) generación 600

Figura 4.13: Proyección del problema de la mochila sobre un SOM entrenado con puntos seleccionados al azar de su espacio de búsqueda, $\mathbb{Z}'$, siguiendo una distribución uniforme de la figura 4.12b. Los números del interior de los gráficos a-f indican el fitness de los individuos proyectados.

que representan bajos valores de fitness a las que se acercan a los óptimos.

- La inicialización de la población, a pesar de ser manifiestamente mejorable, se expande hasta ocupar un 45,63 % del área del mapa, figura 4.13(a), de donde podemos decucir que no es tan mala como anteriormente pudimos deducir de las figuras 4.9(a) y 4.11(a). Sin embargo, como señalamos en la sección 4.2.1, la forma de crear la población inicial debería cambiarse para intentar ocupar la mejor zona del mapa, la zona izquierda, que es donde puede encontrarse algunas de las soluciones óptimas.

- Algo que no se había apreciado en las anteriores proyecciones es que no llegan a explorarse todas las áreas del mapa. La esquina superior-izquierda no es recorrida por el algoritmo durante la búsqueda, con la consiguiente pérdida de oportunidades de encontrar las buenas soluciones que por dicha área pudiesen haber. En la proyección de la figura 4.9 este detalle es imposible de apreciar, pues su SOM está entrenado a partir de individuos obtenidos de ejecuciones del algoritmo, con lo cual las áreas sin explorar simplemente no son representadas. En la proyección de la figura 4.11 aunque sí aparecen dichas áreas inexploradas, dado lo comprimida de su representación, es imposible de apreciar este detalle. Por este motivo, sería útil emplear alguna de las técnicas que permitan ampliar la cantidad de espacio de búsqueda explorada, como, por ejemplo, las de reparto del fitness. En sección 5.2 podemos encontrar una breve descripción de este tipo de técnicas, así como ejemplos de su uso y la aplicación del SOM para su visualización.

4.3. Conclusiones

Como hemos visto a modo de ejemplo para los problemas onemax, rastrigin y mochila, es posible extraer información útil de la representación del espacio de búsqueda creada a partir de los mapas autoorganizativos de Kohonen.

A veces es más útil realizar una aproximación en dos pasos al problema de la visualización: primero utilizando un SOM entrenado con puntos seleccionados de forma aleatoria del espacio de búsqueda completo del problema, y después, si se ha logrado obtener alguna información del primer mapa, utilizar un segundo SOM entrenado utilizando el nuevo conocimiento obtenido acerca del espacio de búsqueda.

Esta información abarca aspectos tales como la calidad de la inicialización de la población, la dirección y velocidad del proceso de convergencia, el estudio del grado de diversidad de la población, la cantidad de espacio de búsqueda explorado, el descubrimiento y diferenciación de la calidad de las diferentes áreas, exploradas o no, del espacio de búsqueda, la forma del espacio de búsqueda o como un algoritmo evolutivo se desenvuelve durante su funcionamiento.

En el caso de usuarios poco experimentados en el mundo de la computación evolutiva puede ser mucho más importante este tipo de técnicas, ya que muchas de las conclusiones a las que puede llegar un usuario avanzado estudiando estadísticas pueden ser más fáciles de obtener a través de este tipo procedimientos visuales.

Además de las posibilidades ya mencionadas, también puede ser útil para satisfacer al simple curiosidad acerca de como funciona un algoritmo evolutivo y poder ver como se desplazan los individuos por el espacio de búsqueda a medida que las generaciones van pasando.

Capítulo 5

Operadores genéticos

Otro de los usos que puede tener la visualización es conocer el efecto de los operadores genéticos y de los valores que damos a sus parámetros. Muchas veces los investigadores añaden o eliminan operadores genéticos mediante el sistema de prueba y error, es decir, añaden un operador y luego ven si las estadísticas mejoran o no. Lo mismo suele ocurrir con los valores de los parámetros. Este método dista mucho de ser la mejor forma de conocer su funcionamiento.

Mediante la visualización se podría descubrir mucha más información útil acerca de ellos, no sólo si funcionan bien o no, sino además, por qué. Otras veces sí se sabe cómo puede afectar un operador a un población pero también es deseable el poder confirmarlo visualmente, especialmente para los que se inician en el campo de la computación evolutiva, o a la hora de probar una nueva implementación de una técnica ya conocida, para confirmar su correcto funcionamiento.

Para estudiar sus efectos en visualización vamos a utilizar un algoritmo evolutivo que entrena redes neuronales artificiales (Yao, 1992). Las redes utilizadas son del tipo perceptrón multicapa (de *Multilayer Perceptron* o *MLP*) y cada neurona utiliza un peso extra denominado sesgo ("*bias*") conectado a una entrada ficticia siempre a 1. En la figura 5.1 podemos ver un ejemplo de este tipo de red neuronal. En (Castillo *et al.* , 2000b) y (Castillo *et al.* , 2000a) se pueden encontrar más detalles acerca de las implementaciones de

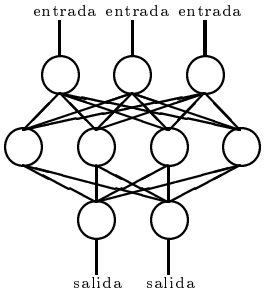


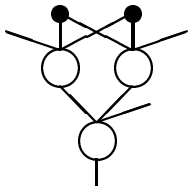
Figura 5.1: Ejemplo de perceptrón multicapa de 3 capas con 3 entradas y 2 salidas. La primera capa tiene 3 neuronas conectadas a las 3 entradas, la capa intermedia tiene 4 neuronas y la capa de salida tiene 2 neuronas.

redes neuronales y algoritmos genéticos utilizadas en los siguientes experimentos.

En primer lugar vamos a estudiar el efecto que produce el uso de dos operados genéticos diferentes sobre la evolución. Los operadores en cuestión son la mutación "tradicional", que cambia valores de los pesos de forma aleatoria (Castillo *et al.* , 2000a), frente a otro tipo de mutación basada en la aplicación del algoritmo quickprop de Falhman (Fahlman, 1988). Por lo demás, el algoritmo evolutivo empleado para entrenar las redes es idéntico. Estos operadores han sido escogidos por tener efectos totalmente opuestos y por considerarse adecuados para estudiar sus efectos sobre la visualización.

5.1. Evolución de redes neuronales

Las redes que emplearemos van a tener un tamaño de 3 neuronas, el mínimo necesario para aprender la función XOR, conectadas según puede verse en la figura 5.2. Cada una de las neuronas tiene 3 pesos, con lo cual el tamaño total de cada red es de 9 pesos. Así pues podemos ver cada red como un vector de $\mathbb{R}^9$. Como en los problemas antes descritos, estas son más dimensiones de las que la visión humana puede abarcar, con lo cual utilizaremos la técnica aquí descrita para intentar visualizar el comportamiento del algoritmo evolutivo que entrena las redes neuronales.



entradas		salida
0	0	0
0	1	1
1	0	1
1	1	0

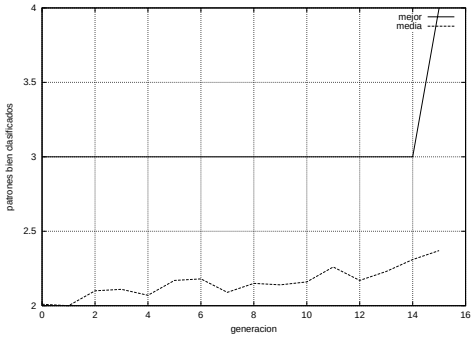
Perceptrón multicapa Función XOR

Figura 5.2: Mínimo perceptrón multicapa capaz de aprender la función XOR. Tiene 2 capas y 3 neuronas. Cada neurona posee un peso extra, denominado sesgo ("*bias*"), conectado a una entrada ficticia cuyo valor siempre es 1. La red en total tiene 9 pesos.

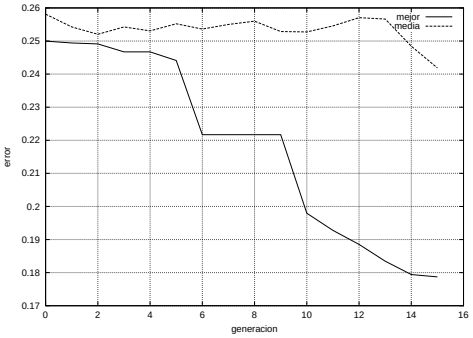
Para este experimento entrenaremos un SOM de 10x10 neuronas con una nube de 262144 puntos escogidos aleatoriamente de la hiperesfera de centro $\mathbf{0}$ y radio 3 de $\mathbb{R}^9$. El tamaño del radio lo escogimos siguiendo algunas pistas dadas por Reed en (Reed & Marks, 1999) y tras observar los valores típicos de los pesos de las redes obtenidas en unas cuantas ejecuciones.

En la figura 5.3 podemos ver el comportamiento del algoritmo cuando se utilizan todos los operadores de forma óptima, es decir, mutación clásica, mutación basada en quickprop y cruce monopunto. El algoritmo encuentra una red que reproduce la función XOR en sólo 15 generaciones. En esta misma figura 5.3, podemos ver las proyecciones de varias etapas de la población sobre el espacio de búsqueda.

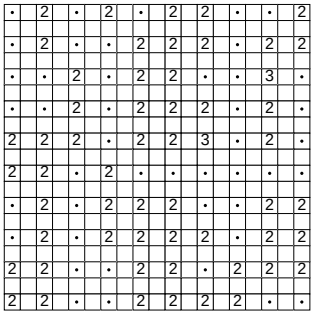
Ahora que hemos visto como se comporta el sistema en condiciones óptimas, vamos a probar a utilizar por separado cada uno de los tipos de mutación. En la figura 5.4 podemos ver el resultado de utilizar sólo la mutación clásica y en la figura 5.5 el resultado de aplicar sólo mutación basada en el algoritmo quickprop. En ambos casos se han mantenido constantes el resto de parámetros así como el cruce monopunto. Como podemos ver en los los tres casos la población inicial es idéntica y sólo se producen diferencias tras las aplicación de los operadores genéticos.



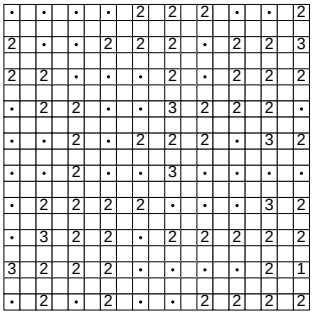
(a) Patrones bien clasificados



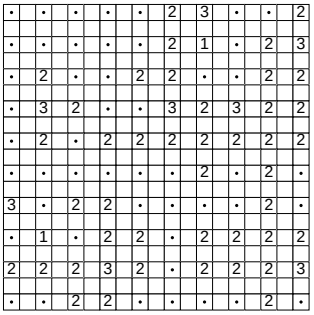
(b) Error cuadrático medio



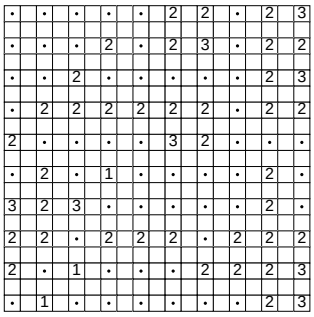
(c) población inicial



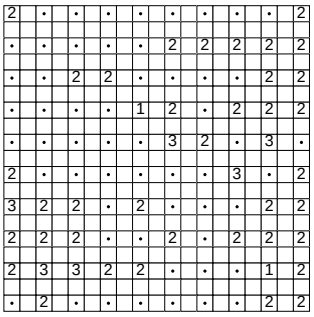
(d) generación 3



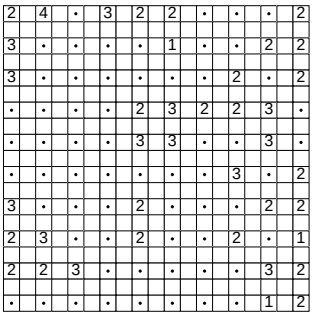
(e) generación 6



(f) generación 9



(g) generación 12



(h) generación 15

Figura 5.3: Estadísticas y proyección del aprendizaje de la función XOR. Los números dentro de los gráficos c-h indican el número de patrones clasificados correctamente. Las etiquetas bajo los gráficos c-h indica el número de generación de la que proceden los datos.

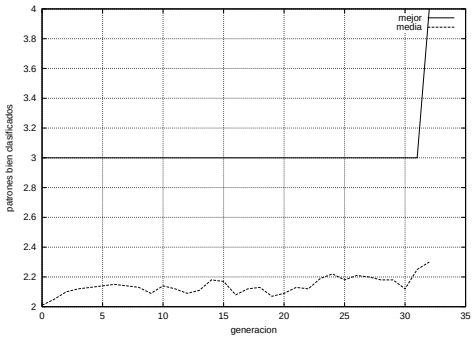
5.1.1. Mutación "clásica"

En el caso de aplicar sólo mutación clásica, el algoritmo se caracteriza por una mejor exploración del espacio de búsqueda, ya que es frecuente la aparición de nuevos individuos en zonas anteriormente vacías, y por tanto inexploradas, como podemos ver en la secuencia de la figura 5.4(c-h). En la figura 5.4a podemos ver como el este tipo de mutación apenas hace mejorar la población ya que es muy difícil que al modificar aleatoriamente una red neuronal se obtenga otra mejor. El error medio incluso aumenta a lo largo de las generaciones, si bien el mejor es cada vez mejor gracias a la selección. En consecuencia podemos apreciar que este operador aumenta la diversidad e introduce buenos bloques constructivos que pueden ser aprovechados por el operador de cruce para dar lugar a mejores individuos. En general esta versión del algoritmo tarda más en encontrar la solución que la que combina los dos tipos de mutación.

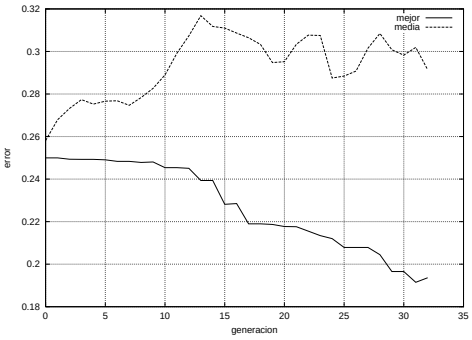
5.1.2. Mutación basada en el algoritmo quickprop

La otra variante, la que sólo emplea mutación basada en el algoritmo quickprop (Fahlman, 1988), tiene unas características opuestas. Se caracteriza por ser más explotador que explorador, ya que cada vez los individuos de la población ocupan menos zonas del mapa, como podemos ver en la figura 5.5(c-h), y a la vez la calidad de los individuos es cada vez mayor. En apenas 20 generaciones la mayoría de los individuos se concentran en muy pocas áreas con lo cual la diversidad de la población se ha visto gravemente afectada. Esta es una de las razones por las que esta versión del algoritmo tarda tanto en encontrar la solución ya que de partir de un buen individuo inicial el encontrar el óptimo está asegurado. Aquí, la única forma de explorar nuevas áreas del espacio de búsqueda es esperar a que el cruce se realice sobre dos individuos suficientemente diferentes como para dar lugar a un descendiente que caiga sobre una nueva zona.

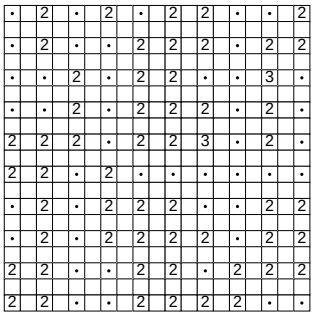
El algoritmo quickprop mejora rápidamente la calidad de las redes, hecho que puede observarse en la figuras 5.5a y 5.5b ya que el número de patrones



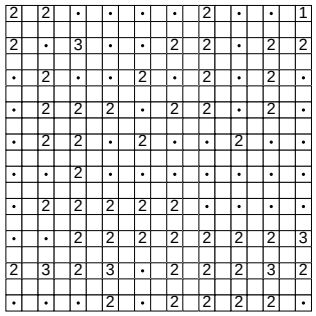
(a) Patrones bien clasificados



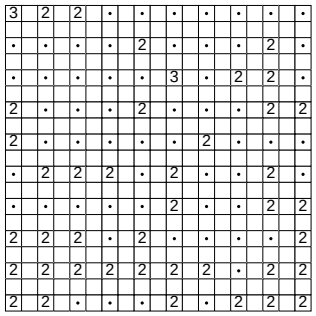
(b) Error cuadrático medio



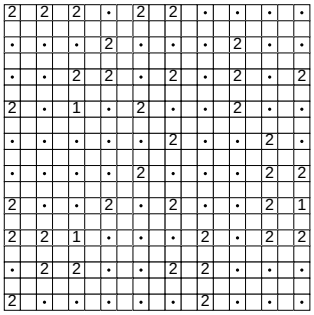
(c) población inicial



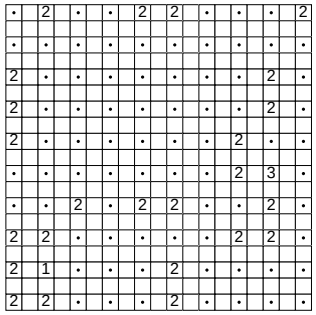
(d) generación 6



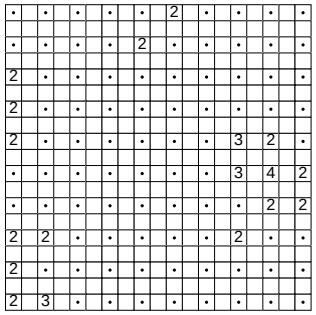
(e) generación 13



(f) generación 19



(g) generación 26



(h) generación 32

Figura 5.4: Estadísticas y proyección del aprendizaje de la función XOR utilizando mutación clásica (ver sección 5.1.1). Los números dentro de los gráficos c-h indican el número de patrones clasificados correctamente. Las etiquetas bajo los gráficos c-h indica el número de generación de la que proceden los datos.

correctamente clasificados crece a la vez que el error cuadrático medio de la población no para de descender. Si comparamos las figuras 5.4a y 5.5a podremos observar que la mutación basada en quickprop hace mejorar la población mucho más deprisa que la mutación clásica. Si ahora comparamos las figuras 5.4b y 5.5b podemos ver que el error cuadrático medio cae de forma sostenida utilizando quickprop, pero no si sólo se emplea mutación aleatoria.

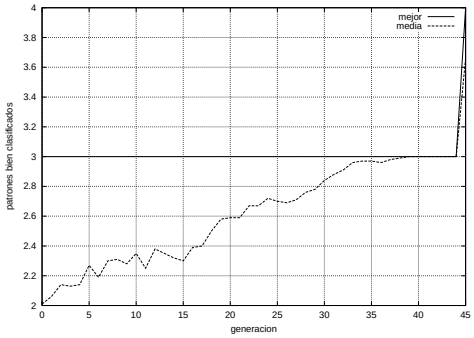
Como hemos visto estos dos operadores de mutación tienen características totalmente opuestas. La mutación clásica tiene la capacidad de introducir en la población nuevos genes, pero no de siempre forma inteligente. Por este motivo, a veces es necesario el concurso de otros operadores para reordenar dichos genes y dar lugar a nuevos y mejores individuos. Su principal efecto es aumentar la diversidad de la población. La otra mutación, basada en quickprop, por el contrario tiene un efecto inmediato sobre la calidad de las soluciones pero en cambio su efecto global es el contrario, reduce la diversidad. Así pues podemos decir que uno de ellos es un operador explorador de nuevas áreas y que el otro en cambio es explotador de las buenas ya encontradas. Es por esto que su combinación obtiene soluciones óptimas de forma eficaz.

Para poner más de manifiesto esta diferencia entre ambos se han entrenado un SOM de mayor tamaño, 20x20 neuronas, sobre los que se han proyectado todos los individuos creados a lo largo de la ejecución los dos algoritmos evolutivos que empleaban un único tipo mutación. En la figura 5.6 vemos que la mutación clásica consigue explorar el 83,25 % del mapa frente al 42,25 % de la mutación basada en quickprop.

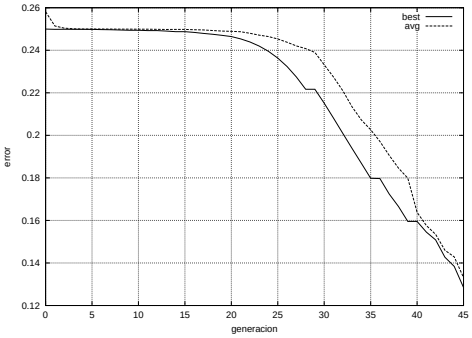
5.2. Técnicas de reparto del fitness

5.2.1. Descripción

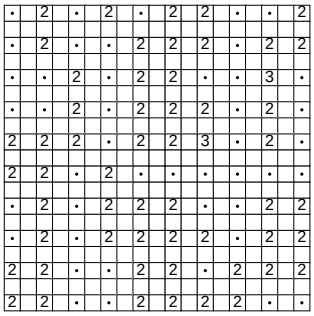
Estas técnicas se basan en tratar el fitness como si fuese un recurso finito por el que los individuos deben competir al igual que sucede en la Naturaleza.



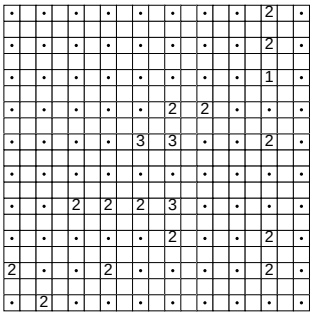
(a) Patrones bien clasificados



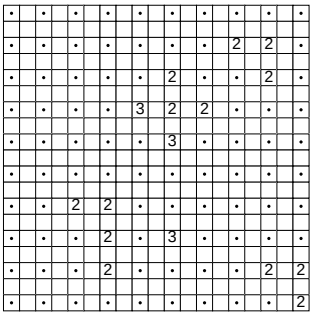
(b) Error cuadrático medio



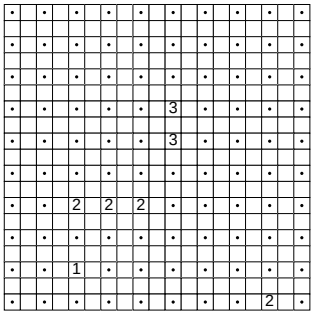
(c) población inicial



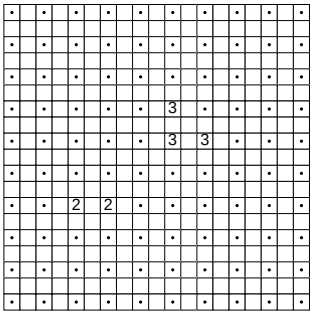
(d) generación 9



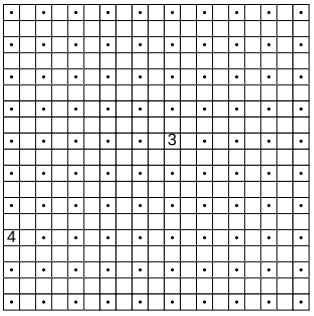
(e) generación 18



(f) generación 27

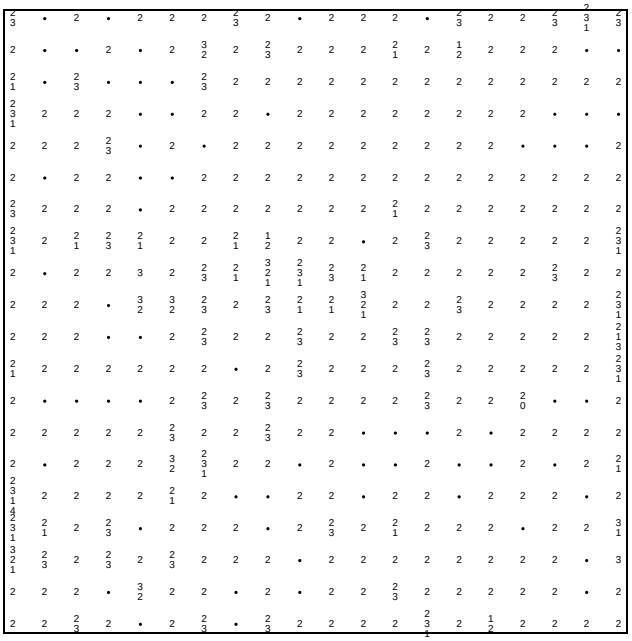


(g) generación 36

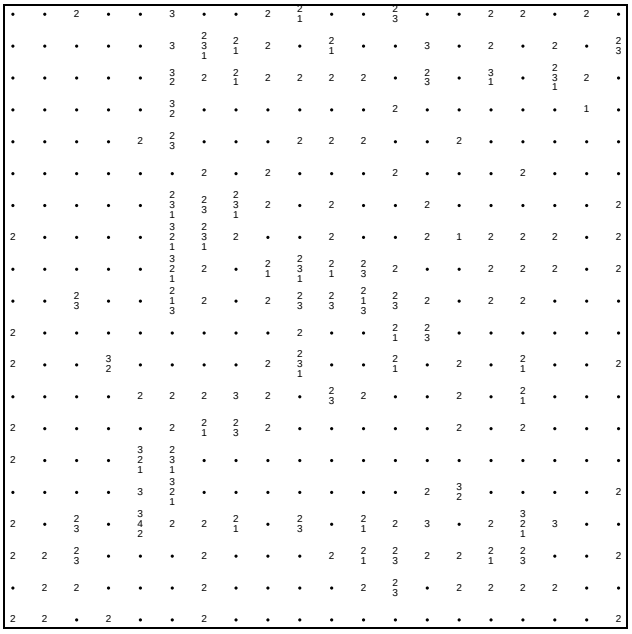


(h) generación 45

Figura 5.5: Estadísticas y proyección del aprendizaje de la función XOR utilizando mutación basada en el algoritmo quickprop (ver sección 5.1.2). Los números dentro de los gráficos c-h indican el número de patrones clasificados correctamente. Las etiquetas bajo los gráficos c-h indica el número de generación de la que proceden los datos.



(a) Área explorada utilizando mutación clásica: 83,25 %



(b) Área explorada utilizando mutación basada en quickprop: 42,25 %

Figura 5.6: Comparativa del área del espacio de búsqueda explorada por un AE al entrenar un perceptrón multicapa para aprender la función XOR empleando dos operadores de mutación diferentes, uno clásico y otro basado en quickprop. En ambos gráficos se han proyectado todos los individuos obtenidos en las dos ejecuciones del algoritmo sobre un mismo SOM.

Para ello los individuos parecidos o cercanos entre sí, ven disminuidos sus valores fitness mientras que los que se encuentran más aislados no se "estorban". Este tipo de técnicas se usan sobre todo en optimización multimodal y multiobjetivo (Coello, 2000). En optimización multimodal es importante localizar todas las soluciones óptimas para un problema. En optimización multiobjetivo se busca optimizar a la vez una serie de subobjetivos. El método que vamos a implementar fue propuesto por Goldberg y Richardson en (Goldberg & Richardson, 1987). La idea es permitir la creación de un conjunto estable de subpoblaciones o nichos mediante la utilización de una función de fitness que degrade el valor de fitness de cada individuo en base a la distancia a que se encuentra de sus vecinos. Una función así, sh, debe cumplir que:

- $0 \leq sh(d) \leq 1 \quad$ para cualquier distancia $\quad d$
- $sh(0) = 1$
- $lim_{d \to \infty} sh(d) = 0$

De las muchas funciones que cumplen las condiciones anteriores utilizaremos la sugerida por Goldberg en (Goldberg & Richardson, 1987):

$$sh(d) = \begin{cases} 1 - (\frac{d}{\sigma_{sh}})^{\alpha} & \text{si} \quad d < \sigma_{sh} \\ 0 & \text{en otro caso} \end{cases} \tag{5.1}$$

donde σ_{sh} y α con constantes que determinan el grado de competencia entre vecinos y el radio de vecindad (para más detalles ver (Goldberg & Richardson, 1987)).

La nueva función de fitness (repartido), f' de un individuo x viene dada por la función:

$$f'(x) = \frac{f(x)}{m(x)} \tag{5.2}$$

donde $f(x)$ es la función de fitness original y $m(x)$ es la suma de todas las distancias desde el individuo x al resto de individuos de la población, o, la

suma del nicho en que x está:

$$m(x) = \sum_{y} sh(d(x,y)) \tag{5.3}$$

El uso de las ecuaciones 5.1 a 5.3 implica que cuando varios individuos están situados cerca entre sí, se incremente su valor de reparto del fitness, con lo cual disminuyen mutuamente sus valores de fitness. De esta forma se evita el crecimiento descontrolado de una determinada especie dentro de una población, o lo que es lo mismo, la acumulación de individuos en un mismo nicho o área del espacio de búsqueda.

Para poner a prueba esta técnica y su interacción con el método de visualización propuesto en esta tesis, vamos a realizar dos series de experimentos. Volveremos a utilizar el algoritmo evolutivo para entrenar redes neuronales, pero esta vez intentaremos que generalice sobre dos conjuntos de datos de mayor dificultad que los utilizados previamente. En el primero de ellos, cáncer, la red debe distinguir tumores cancerígenos benignos y malignos (sección 5.2.2). En el segundo, sonar, la red debe aprender a discriminar si el rebote del eco de un sónar procede de una roca o de un cilindro de metal (sección 5.2.3).

5.2.2. Clasificación de tumores de cáncer de pulmón

Este problema consiste en diagnosticar como benigno o maligno las muestras de un tumor de cáncer de pulmón a partir de la descripción de células analizadas al microscopio. Este conjunto de datos fue recopilado por el Dr. William H. Wolberg en el Hospital Universitario de Winsconsin, ver (Mangasarian *et al.* , 1990) para más detalles, y podemos obtenerlo en el repositorio de pruebas para aprendizaje automático del UCI (Blake & Merz, 1998). Prechelt realiza un exhaustivo análisis del este conjunto de datos en (Prechelt, 1994)

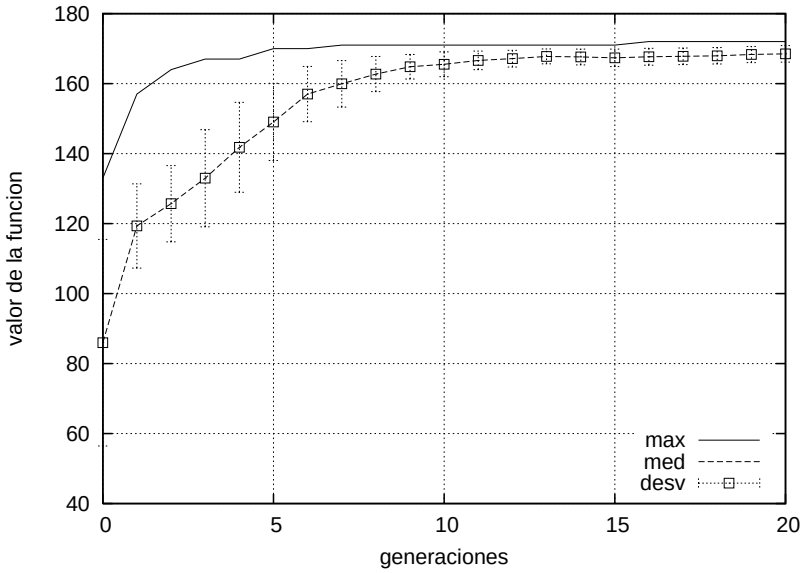
Cada patrón tiene 10 atributos más la clase a la que pertenece. Los atributos son: código de patrón, grosor del grupo de células, uniformidad

del tamaño de las células, uniformidad de la forma de las células, adhesión marginal, tamaño del epitelio celular, núcleos desnudos, suavidad cromática, núcleos normales y mitosis. Las clases posible son dos: benigno o maligno. La distribución de las clases es del 65.5 % de patrones de tumores benignos y del 34.5 % restante de tumores malignos.

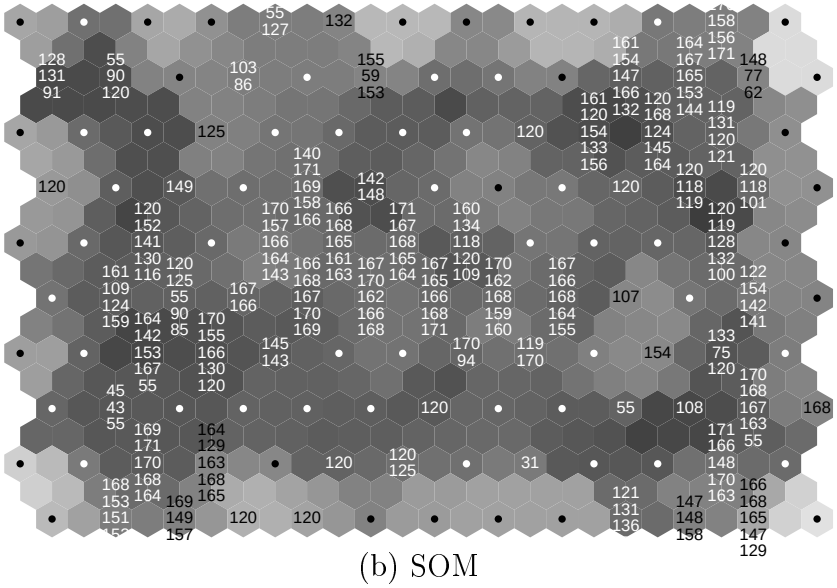
Existen varias particiones del conjunto de datos original. En los siguientes experimentos hemos empleado el denominado `cancer1a`, que es el utilizado por Grönroos (Grönroos, 1998) y Castillo (Castillo, 2000) de los tres propuestos por Prechelt en (Prechelt, 1994).

Como el fin que se persigue aquí no es que la red se capaz de reconocer todos los patrones presentes en los conjuntos de entrenamiento, sino que sea capaz de generalizar ante nuevas entradas desconocidas, utilizaremos una técnica de entrenamiento denominada validación cruzada, ver (Reed & Marks, 1999). Esta técnica consiste en dividir el conjunto de datos inicial en tres subconjuntos. Dos de ellos se utilizan durante la evolución de las redes neuronales, uno de ellos para entrenarlas y otro para validar su capacidad generalizadora. El tercero es utilizado únicamente tras finalizar el algoritmo evolutivo para comprobar la calidad de las redes obtenidas y se le denomina conjunto de test.

El número de entradas utilizadas ha sido de 9 y se ha empleado una red neuronal monocapa con una única neurona de salida con 10 pesos (uno por entrada más el peso de sesgo). Con lo cual las redes neuronales obtenidas como solución a este problema pertenecen al conjunto $\mathbb{R}^{10}$. En este caso se ha entrenado el SOM con 10000 puntos escogidos al azar siguiendo un distribución uniforme de la hiperesfera de centro 0 y radio 2 de $\mathbb{R}^{10}$. En las figuras 5.7 y figuras 5.8, y 5.9 y 5.10 podemos ver el resultado de la proyección de dos ejecuciones de dos algoritmos evolutivos para buscar la solución del problema de la mochila. En el primer caso se ha empleado un algoritmo genético clásico y el segundo uno igual al anterior salvo por la utilización del reparto del fitness. En ambos casos, el SOM empleado para la proyección ha sido el mismo como puede verse en las figuras 5.7 y 5.9. Como podemos apreciar en las figuras 5.8(a) y 5.10(a), ambas ejecuciones parten de la misma

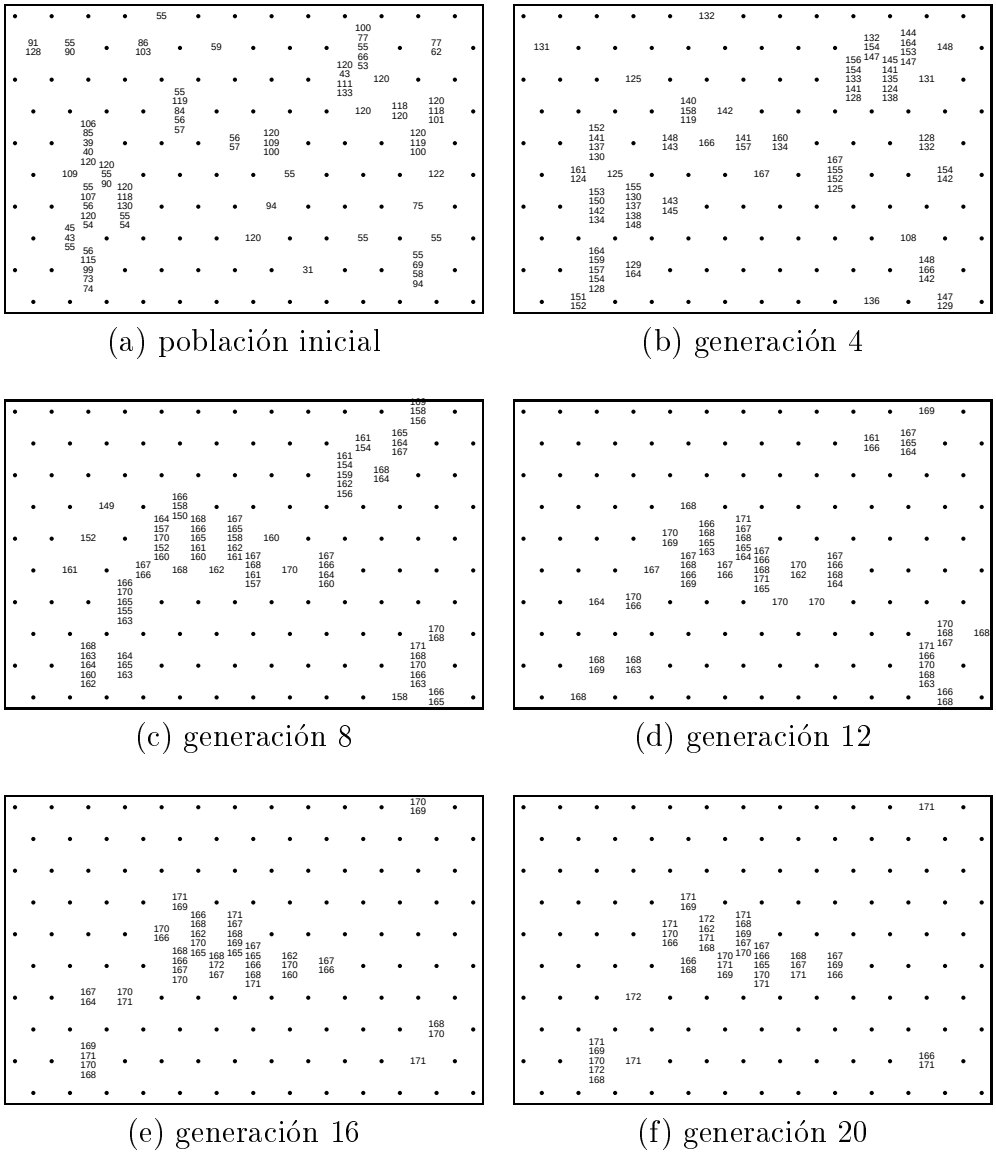


(a) estadísticas



(b) SOM

Figura 5.7: SOM para proyección del problema cancer sin utilizar reparto del fitness sobre un SOM entrenado con puntos seleccionados al azar de la hiperesfera de centro 0 y radio 2 de $\mathbb{R}^{10}$, siguiendo una distribución uniforme. (a) Estadísticas de la ejecución cuya proyección puede verse en la figura 5.8. (b) SOM con el que se ha creado la proyección de la figura 5.8, los números indican el valor de fitness de los individuos proyectados en cada neurona.



(a) población inicial (b) generación 4

(c) generación 8 (d) generación 12

(e) generación 16 (f) generación 20

Figura 5.8: Proyección del problema cancer sin utilizar reparto del fitness sobre el SOM entrenado con puntos seleccionados al azar de la hiperesfera de centro 0 y radio 2 de $\mathbb{R}^{10}$, siguiendo una distribución uniforme de la figura 5.7. Las etiquetas bajo los gráficos indican la generación de la que proceden los individuos proyectados, y los números dentro de los mismo, su valor de fitness.

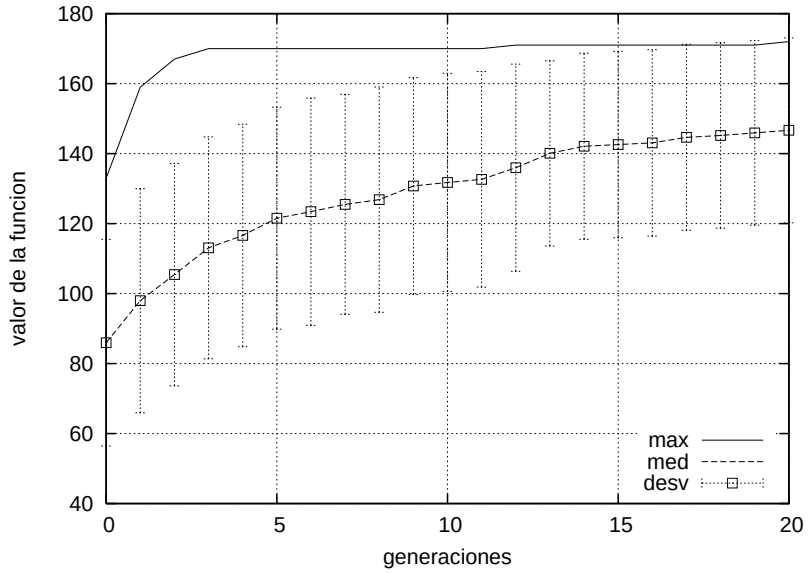
población inicial, pero el uso del reparto del fitness hace que se comporten de forma diferente a medida que avanza el proceso evolutivo.

El efecto más claro, y deseado, del reparto del fitness es el mantenimiento de la diversidad en la población. Esta cualidad podemos apreciarla de varias maneras. La primera, empleada habitualmente por los usuarios de este tipo de algoritmos, es fijarse en la diferencia existente entre las figuras 5.8(a) y 5.10(a). Como podemos ver en ellas la desviación típica disminuye rápidamente cuando no repartimos el fitness, figura 5.8(a), y en cambio se mantiene constante cuando si lo hacemos, figura 5.10(a). La otra forma de hacerlo, empleando el método propuesto en esta tesis, es observar la diferencia entre las secuencias que muestran como ocurre el proceso de convergencia en las figura 5.8 y 5.10. En la primera de ellas podemos ver como a medida que avanza la evolución, la población va concentrándose en ciertas regiones del espacio de búsqueda, en concreto la zona central del mapa, ver figura 5.8(f), hasta ocupar un 10,77 % de sus celdas. En la segunda en cambio el proceso de convergencia parece no ocurrir nunca e incluso al final, 5.10(f), la población aparece repartida por la mayor parte del mapa cubriendo el 33,85 % de sus celdas. En ambas ejecuciones, al partir de idénticas condiciones iniciales, las poblaciones ocupan el 24,62 % de las celdas del mapa. Como vemos, el empleo del reparto del fitness no sólo ayuda a mantener la diversidad de la población, sino que la aumenta. La utilización del SOM permite apreciar visualmente dicho aumento de la diversidad y además proporciona una forma rápida de medirla calculando el área ocupada.

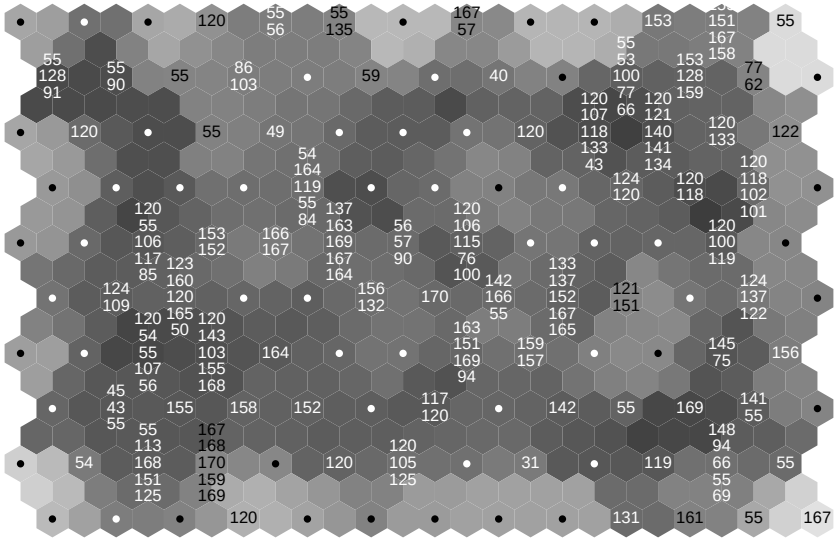
5.2.3. Clasificación de ecos de sónar

Para estudiar los efectos de las técnicas de reparto del fitness de nuevo emplearemos un algoritmo evolutivo para entrenar redes neuronales, pero esta vez se ha escogido un problema de mayor dificultad de reconocimiento de patrones propuesto por Gorman y Sejnowski en (Gorman & Sejnowski, 1988). Para un estudio más exhaustivo de este problema ver (Castillo, 2000).

Este problema, denominado sonar, consiste en el reconocimiento de ecos

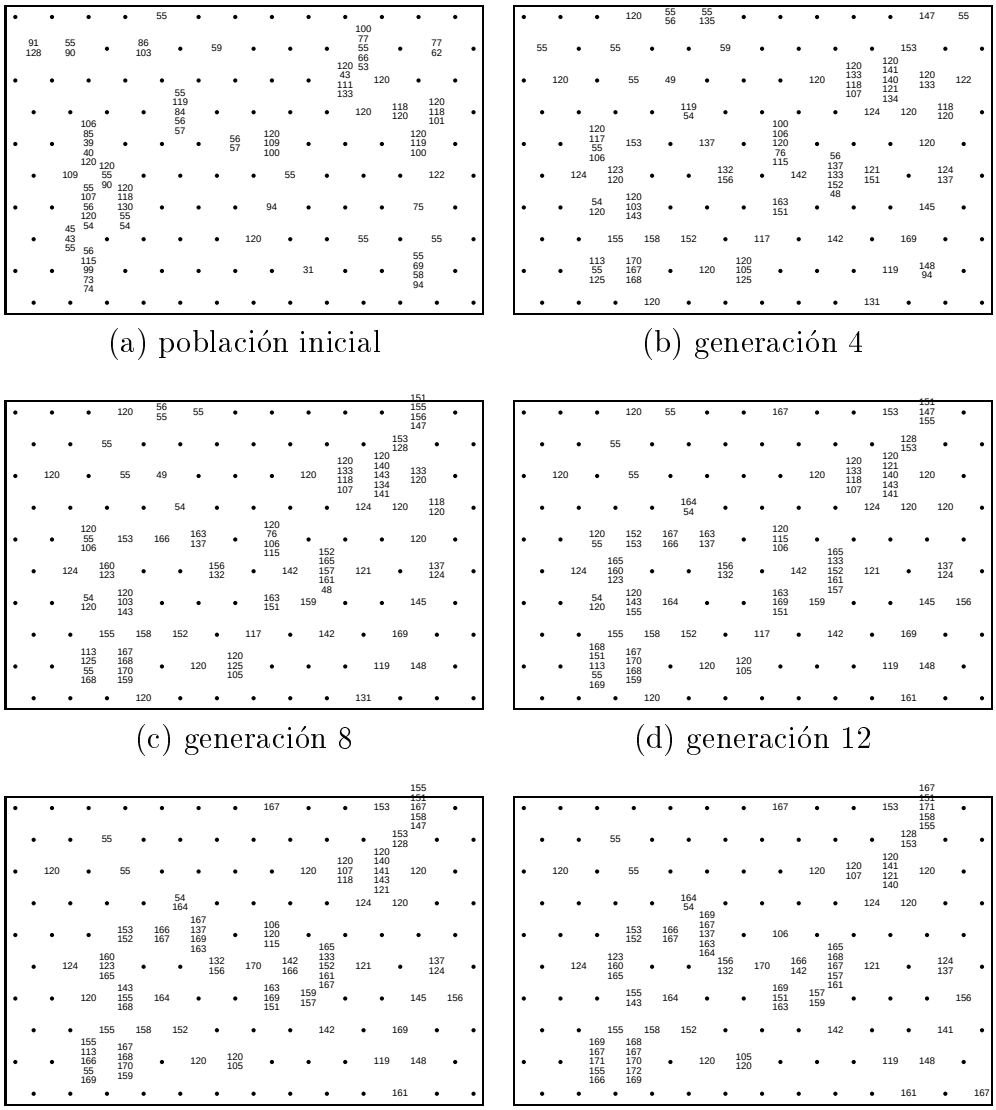


(a) estadísticas



(b) SOM

Figura 5.9: SOM para Proyección del problema cancer utilizando reparto del fitness sobre un SOM entrenado con puntos seleccionados al azar de la hiperesfera de centro 0 y radio 2 de $\mathbb{R}^{10}$, siguiendo una distribución uniforme. (a) Estadísticas de la ejecución cuya proyección puede verse en la figura 5.10. (b) SOM con el que se ha creado la proyección de la figura 5.10, los números indican el valor de fitness de los individuos proyectados en cada neurona.



(a) población inicial (b) generación 4

(c) generación 8 (d) generación 12

(e) generación 16 (f) generación 20

Figura 5.10: Proyección del problema cancer utilizando reparto del fitness sobre el SOM entrenado con puntos seleccionados al azar de la hiperesfera de centro 0 y radio 2 de $\mathbb{R}^{10}$, siguiendo una distribución uniforme que puede verse en la figura 5.9. Las etiquetas bajo los gráficos indican la generación de la que proceden los individuos proyectados, y los números dentro de los mismo, su valor de fitness.

de sónar tras reflejarse en dos tipos de objetos diferentes: un cilindro de metal, calificado como mina, y una roca de forma cilíndrica posicionada longitudinalmente en el suelo marino. El pulso enviado era una señal de banda ancha lineal de frecuencia modulada. Los ecos fueron obtenidos desde diversos ángulos para cada objetivo.

Hay un total de 208 ecos, 111 de ellos procedentes del cilindro metálico y 97 de la roca, seleccionados en función de su relación señal/ruido. De ellos 104 son apartados para entrenamiento, 52 para validación y 52 para test, equilibrando la cantidad de minas y rocas presentes en la muestra. La señal reflejada por el objetivo es sometida a una transformada de Fourier y posteriormente se calcula su envolvente espectral. De cada envolvente espectral se obtienen 60 muestras que son normalizadas en el rango 0.0 a 0.1. Cada número representa un nivel de energía dentro de una banda de frecuencias particular.

Este problema forma parte del conjunto de datos de prueba para redes neuronales de la Universidad Carnegie Mellon y está disponible en (Blake & Merz, 1998).

Al igual que en la sección anterior, como nuestro objetivo no es que la red memorice el conjunto de entrenamiento perfectamente, sino sea capaz de generalizar, entonces también utilizaremos validación cruzada.

Para poder observar mejor las propiedades de las técnicas de reparto del fitness ejecutaremos dos veces un mismo experimento en el que permanecerán constantes todos los parámetros salvo por el uso de esta técnica en el segundo de ellos. También los proyectaremos sobre un mismo SOM para poder compararlos mejor.

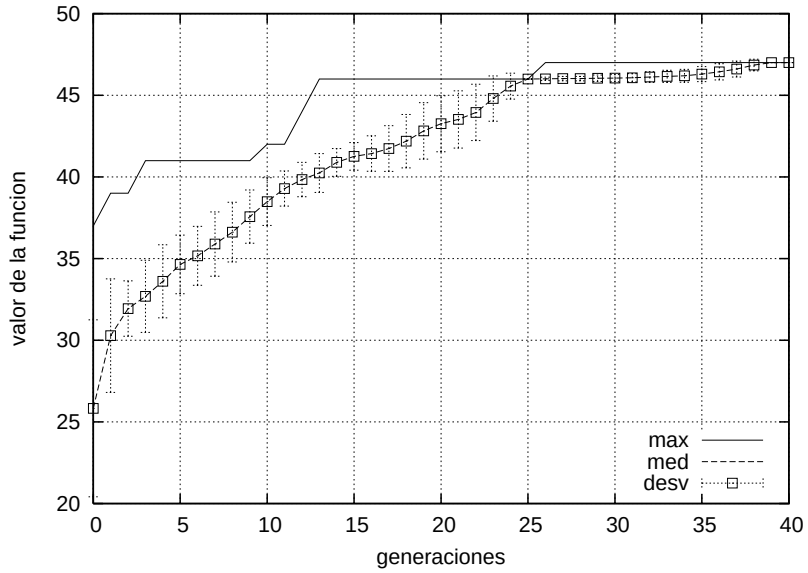
Cada red neuronal está formada por 3 neuronas y 125 pesos, con lo que nuestro espacio de búsqueda ahora está dentro de $\mathbb{R}^{125}$. Teniendo el tamaño de las redes en cuenta, se entrenó un SOM con 10000 puntos obtenidos al azar según una distribución uniforme de la hiperesfera de centro 0 y radio 4 de $\mathbb{R}^{125}$. Al igual que hicimos en la sección anterior, ejecutaremos dos experimentos, con y sin reparto del fitness, y los proyectaremos sobre un mismo SOM para poder comparar mejor sus efectos. Los resultados podemos

verlos en las figuras 5.11 y 5.12, y 5.13 y 5.14.

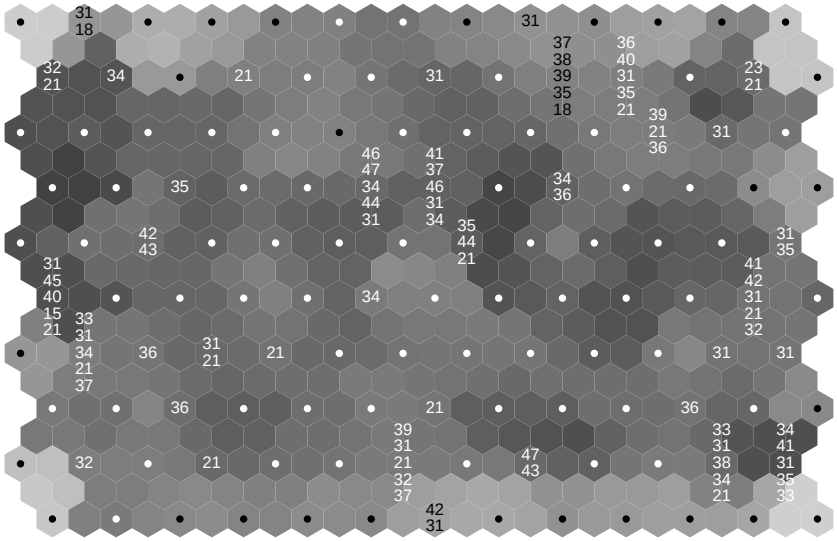
Estudiando la ejecución en que se utiliza un algoritmo genético simple, figure 5.12, vemos que la cantidad de espacio de búsqueda explorada es mínima. La población inicial sólo consigue cubrir el 19,23 % de la celdas del SOM, ver figura 5.12(a), y a medida que avanzan las generaciones esta cantidad disminuye hasta llegar a concentrar a toda la población en dos única celdas en la generación 40, ver figura 5.12(f), lo que supone e. 1,54 % del área total. Una información parecida puede obtenerse en la figura 5.11(a) sin más que observar como disminuye la desviación típica de la cantidad de patrones correctamente clasificados. Sin embargo esta segunda forma sólo nos informa de la diversidad en el valor de las soluciones, y no de la estructura de las mismas, ya que podriamos tener soluciones de diferente estructura (redes neuronales con diferentes conjuntos de pesos) y de igual poder de clasificación. Este fenómeno, que realmente sí ocurre en esta ejecución del algoritmo, no se podría apreciar simplemente fijándonos en sus valores de fitness, ver figura 5.11(a). En cambio, en la figura 5.12(f), sí podemos apreciar que existen dos tipos de soluciones de igual calidad pero de diferente estructura.

Analizando ahora la figura 5.14 podemos apreciar los cambios producidos en la evolución por el uso de la técnica de reparto del fitness. La diversidad inicial es la misma que en la ejecución anterior, ver figura 5.12(a), ya que partimos de las mismas condiciones iniciales. Dicha población inicial ocupaba el 19,23 % de área total del SOM. A diferencia de lo que ocurria antes, la diversidad, en vez de de disminuir, aumenta hasta hacer que la población ocupe el 25,38 % de las celdas del mapa en la última generación, ver figura 5.14(f). Como en el caso de conjunto de datos cancer (sección 5.2.2), se cumple el objetivo para el que hemos empleado la técnica de reparto del fitness, el mantenimiento de la diversidad de la población. Dicha diversidad se mantiene no sólo en la calidad de las soluciones, ver figura 5.13(a), sino también en las estructuras de las mismas, ver figura 5.14(f).

Como efecto negativo, destacar que al aumentar la cantidad de espacio explorado, la explotación de buenas soluciones existentes en la población no puede ser igual de eficaz. El algoritmo divide su atención en muchos más

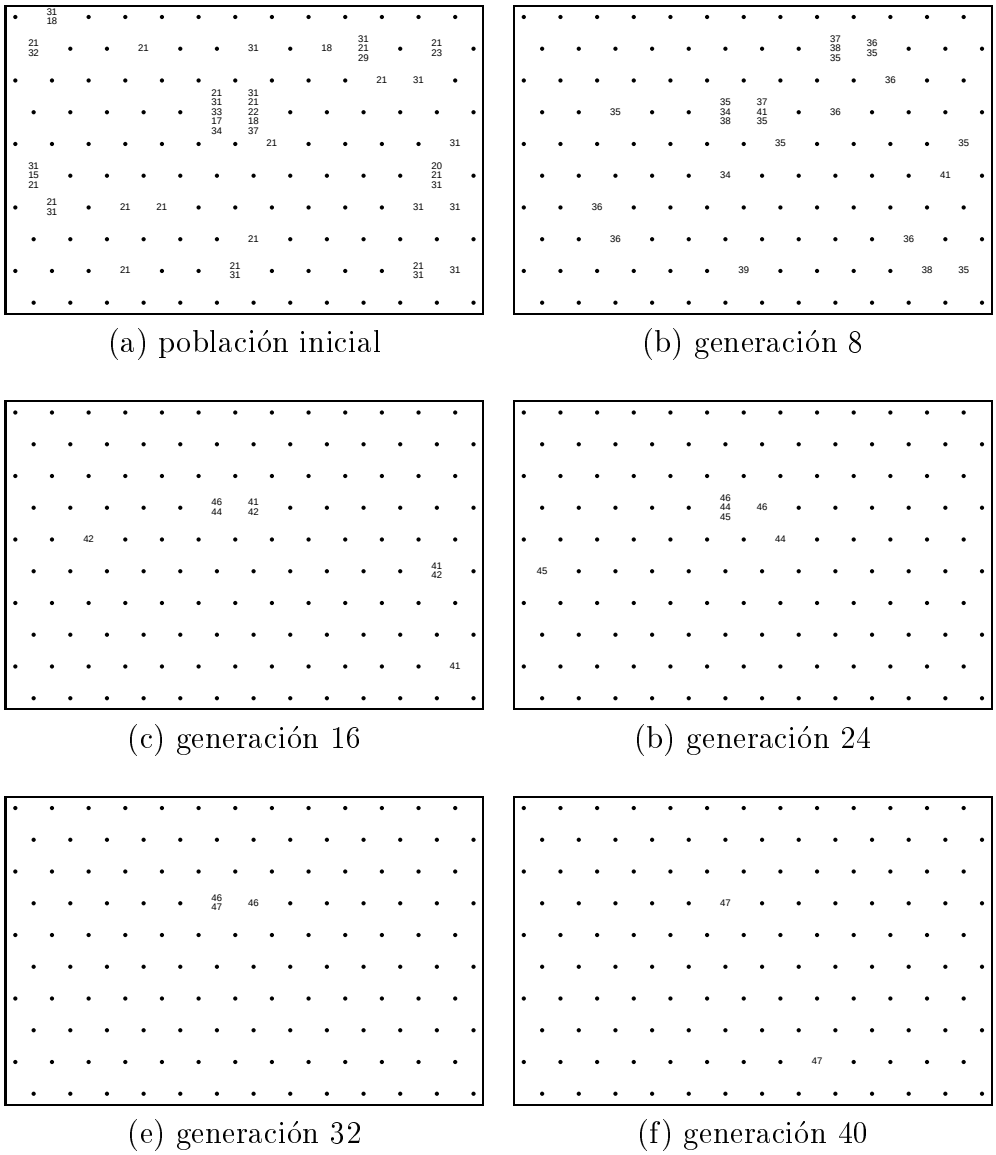


(a) estadísticas



(b) SOM

Figura 5.11: SOM para proyección del problema sonar sin utilizar reparto del fitness sobre un SOM entrenado con puntos seleccionados al azar de la hiperesfera de centro 0 y radio 4 de $\mathbb{R}^{125}$, siguiendo una distribución uniforme. (a) Estadísticas de la ejecución cuya proyección puede verse en la figura 5.12. (b) SOM con el que se ha creado la proyección de la figura 5.12, los números indican el valor de fitness de los individuos proyectados en cada neurona.

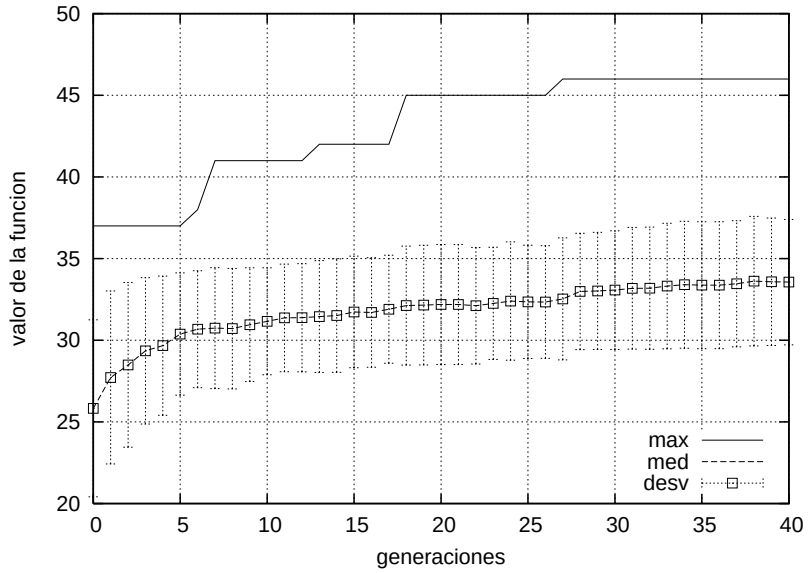


(a) población inicial (b) generación 8

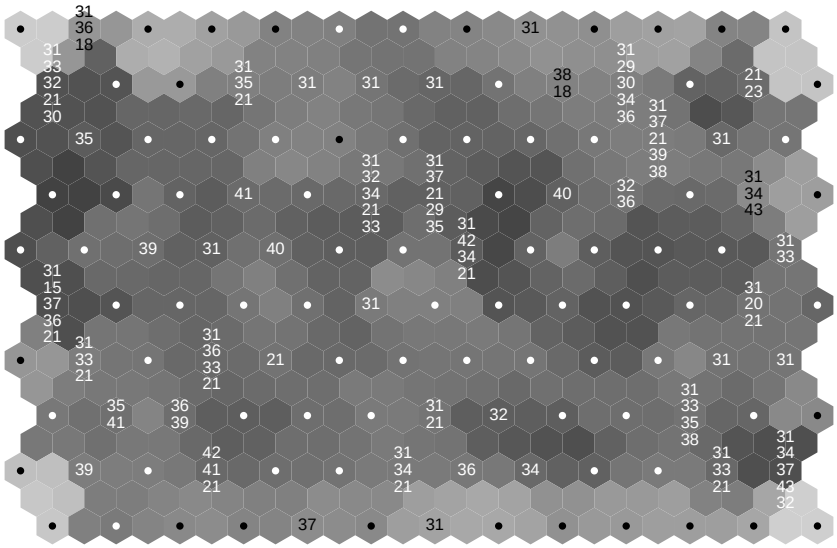
(c) generación 16 (b) generación 24

(e) generación 32 (f) generación 40

Figura 5.12: Proyección del problema sonar sin utilizar reparto del fitness sobre el SOM entrenado con puntos seleccionados al azar de la hiperesfera de centro 0 y radio 4 de $\mathbb{R}^{125}$, siguiendo una distribución uniforme que puede verse en la figura 5.11. Las etiquetas bajo los gráficos indican la generación de la que proceden los individuos proyectados, y los números dentro de los mismo, su valor de fitness.

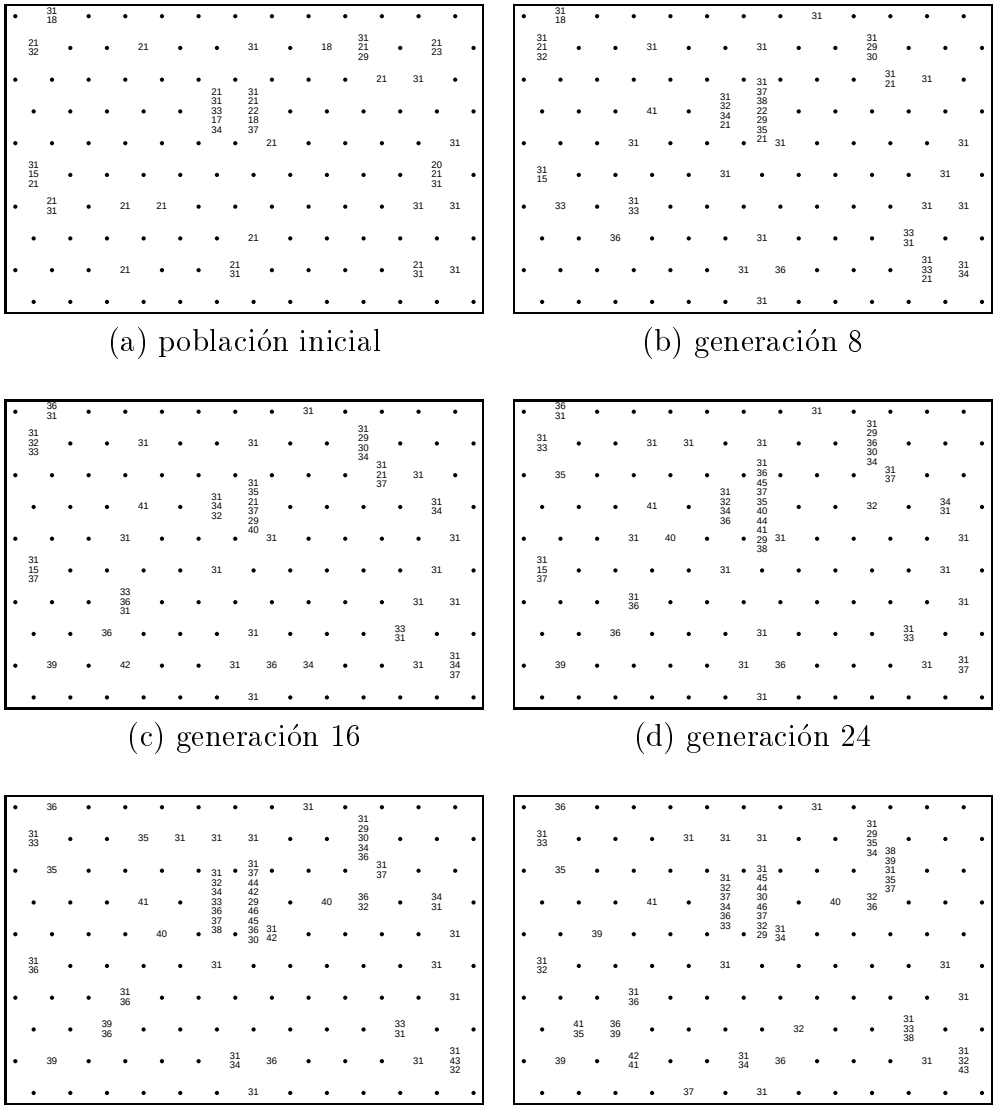


(a) estadísticas



(b) SOM

Figura 5.13: SOM para proyección del problema sonar utilizando reparto del fitness sobre un SOM entrenado con puntos seleccionados al azar de la hiperesfera de centro 0 y radio 4 de $\mathbb{R}^{125}$, siguiendo una distribución uniforme. (a) Estadísticas de la ejecución cuya proyección puede verse en la figura 5.14. (b) SOM con el que se ha creado la proyección de la figura 5.14, los números indican el valor de fitness de los individuos proyectados en cada neurona.



(a) población inicial (b) generación 8

(c) generación 16 (d) generación 24

(e) generación 32 (f) generación 40

Figura 5.14: Proyección del problema sonar utilizando reparto del fitness sobre el SOM entrenado con puntos seleccionados al azar de la hiperesfera de centro 0 y radio 4 de $\mathbb{R}^{125}$, siguiendo una distribución uniforme que puede verse en la figura 5.13. Las etiquetas bajo los gráficos indican la generación de la que proceden los individuos proyectados, y los números dentro de los mismo, su valor de fitness.

frentes y de esta forma la mejor solución encontrada es ligeramente inferior a la obtenida anteriormente. Pero a cambio se gana en la cantidad de soluciones encontradas.

El usuario de este tipo de algoritmos será el que en cada caso evalúe que le es más valioso, si encontrar muchas soluciones diferentes o centrarse en encontrar una única solución lo mejor posible.

5.3. Conclusiones

En este capítulo se ha visto como los efectos de los operadores genéticos se ponen de manifiesto de una forma visual al proyectar las poblaciones obtenidas en cada generación sobre un SOM. Esto ha quedado patente en la sección 5.1 en la diferente forma de explorar el espacio de búsqueda de un mismo problema por parte de dos operadores de variación de individuos diferentes: una mutación aleatoria clásica y una mutación basada en el algoritmo quickprop.

En segundo lugar se han mostrado, también de forma visual, los efectos de la utilización de dos tipos de algoritmos diferentes dentro del campo de la computación evolutiva, como son el Algoritmo Genético Simple (SGA) y los Algoritmos Genéticos con Reparto del Fitness. Como se puede ver en los ejemplos de la sección 5.2, la principal diferencia entre ambos es la cantidad de espacio de búsqueda explorado y el mantenimiento de la diversidad de la población, que son claramente mejores en el caso de utilizar reparto del fitness, a costa de un mayor coste computacional.

Como se dijo anteriormente, si bien en estos casos ya es conocido teóricamente el funcionamiento de los operadores y algoritmos empleados, sin utilizar alguna técnica como la aquí propuesta no seríamos capaces de corroborarlo visualmente. Además, a la hora de probar nuevos algoritmos, pueden darnos información útil acerca de su funcionamiento, ya que en el campo de la computación evolutiva es muy difícil predecir teóricamente el comportamiento de la evolución de una población.

Capítulo 6

Conclusiones y trabajo futuro

En el trabajo de tesis doctoral presentado en esta memoria se han aunado varios métodos computacionales como la computación evolutiva y los métodos de escalado multidimensional, con el objetivo de comprender y mejorar el funcionamiento de los primeros.

Las limitaciones de la visión humana en cuanto al número de dimensiones que podemos apreciar de forma simultanea hace necesario emplear algún tipo de técnica de escalado multidimensional que reduzca la dimensionalidad de los datos en aquellos problemas que lo requieran y nos permita visualizarlos. De todas estas técnicas enumeradas en la sección 3.2, el Mapa Autoorganizativo de Kohonen es la que mejor se adapta a nuestros intereses por sus buenas características de proyección para problemas de alta dimensionalidad, por ser la única técnica que permite escalar y visualizar a la vez, por proyectar cualquier punto una vez entrenado, y por la amplia base de software existente, que hace que esté al alcance de cualquier investigador, incluso de forma gratuita.

Para una óptima visualización es crítico el conjunto de datos con el que el SOM es entrenado. Cualquier pista sobre la forma del espacio de búsqueda de un problema nos ayudará a escoger este de forma lo más precisa posible para así obtener buenas proyecciones. En cualquier caso, este nunca es un proceso de búsqueda a ciegas ya que siempre podemos entrenar un mapa con una representación de puntos aleatoriamente seleccionados del espacio

de búsqueda del problema, y después, refinarlo basándonos en las zonas del mapa que explora el algoritmo evolutivo que lo resuelve. En los capítulos 4 y 5 podemos encontrar ejemplos sobre cómo escoger los conjuntos de entrenamiento para el SOM y sobre como refinar dicho conjunto a partir de alguna otra pista acerca de un problema.

Una vez obtenido un mapa adecuado este puede ser empleado para obtener proyecciones de las que extraer una gran cantidad de información valiosa acerca de un algoritmo evolutivo, tal como: si la población inicial es creada de forma adecuada, que grado de diversidad existe en la población inicial y como se mantiene este a lo largo de la evolución, cómo y cuándo ocurre el fenómeno de la convergencia, si este tiene lugar, si la búsqueda encuentra una única solución o varias de ellas, como afecta un operador genético a la manera en que los individuos exploran el espacio de búsqueda, la forma del espacio de búsqueda, o el porcentaje de espacio explorado.

En el caso de usuarios poco experimentados en el mundo de la computación evolutiva puede ser mucho más importante este tipo de técnicas, ya que muchas de las conclusiones a las que puede llegar un usuario avanzado estudiando estadísticas pueden ser más fáciles de obtener a través de este tipo procedimientos visuales.

Además de las posibilidades ya mencionadas, también puede ser útil para satisfacer al simple curiosidad acerca de como funciona un algoritmo evolutivo y poder ver como se desplazan los individuos por el espacio de búsqueda a medida que las generaciones van pasando.

En el caso de utilizar algoritmos de funcionamiento teóricamente ya conocido, sin la utilización de técnicas de visualización como la aquí propuesta no seríamos capaces de corroborarlo visualmente. Además, a la hora de probar nuevos algoritmos, pueden darnos información útil acerca de su funcionamiento, ya que en el campo de la computación evolutiva es muy difícil predecir teóricamente el efecto de un algoritmo sobre una población.

A lo largo de la realización de esta tesis se han ido obteniendo diversos resultados que han quedado reflejados en las siguientes publicaciones:

- G. Romero; M.G. Arenas; J.G. Castellano; P.A. Castillo; J. Carpio;

J.J. Merelo; A. Prieto; V.M. Rivas. Evolutionary Computation Visualization: Application to G-Prop. Pages 902-912 of Proceedings of the 6th Parallel Problem Solving from Nature - PPSN VI. Lecture Notes in Computer Science, no. 1917. Springer-Verlag. 2000.

- P.A. Castillo; J. Carpio; J.J. Merelo; V.M. Rivas; G. Romero; A. Prieto. Evolving Multilayer Perceptrons. Neural Procesing Letters, 12(2), 115–127. 2000.

- G. Romero; P.A. Castillo; J.J. Merelo; A. Prieto. Using SOM for Neural Network Visualization. Pages 629-636 of Proceedings of the 6th International Conference on Artificial Neural Networks (IWANN'2001). Lecture Notes in Computer Science, no. 2084. Springer-Verlag. June 2001.

- M. Keijzer; J.J. Merelo; G. Romero; M. Schoenauer. Evolving Objects: a general purpose evolutionary computation library. Pages 442-451 of Proceedings of Evolution Artificielle - EA'2001. Lecture Notes in Computer Science, no. 2310. Springer-Verlag. 2001.

- I. Rojas; J. González; H. Pomares; J.J. Merelo; P.A. Castillo; G. Romero. Statistical Analysis of the Main Parameters Involved in the Desing of a Genetic Algorithm. IEEE Transactions on Systems, Man and Cybernetics, 32(1), 31–37. IEEE Press. 2002.

- G. Romero; J.J. Merelo; P.A. Castillo; J.G. Castellano; M.G. Arenas. Genetic Algorithm Visualization using Self-Organizing Maps. Proceedings of the 7th Parallel Problem Solving from Nature - PPSN VII. Lecture Notes in Computer Science, no. 2439. Springer-Verlag. September 2002.

- P.A. Castillo; J.J. Merelo; A. Prieto; I. Rojas; G. Romero. Statistical Analysis of the parameters of a neuro-genetic algorithm. IEEE Transactions on Neural Networks, 13(6), 1374–1394. IEEE Press. 2002.

Los resultados presentados en este trabajo constituyen un punto de partida para un estudio más completo tanto del método de visualización propuesto como de las aplicaciones para las que es adecuado. Algunas de las lineas de trabajo futuro son las que se proponen a continuación:

- Sería interesante probar otras técnicas de escalado multidimensional, tales como el Análisis de Componentes Curvilíneos o el Análisis de Distancias Curvilíneas, pues para ciertos tipos de espacios de búsqueda altamente no lineales podrían producir mejores proyecciones, además de ser computacionalmente más ligeros. El segundo puede ser especialmente interesante en el caso de espacios de búsqueda que sean mejor caracterizados por distancias de tipo curvilíneas en vez de la clásica distancia euclídea.

- Disponer de una herramienta que automatizase el proceso de visualización y lo aplicase de forma interactiva y a través de un interfaz gráfico ayudaría a expandir este método de visualización entre la comunidad científica. Actualmente todo el proceso se realiza a base de pequeños programas sin interfaz gráfico.

- Se podría crear operadores basados en el SOM para permitir explorar áreas no visitadas del espacio de búsqueda.

- Otra utilidad del SOM sería emplearlo para evitarnos el tener que calcular las distancias entre cada par de individuos en las técnicas de reparto del fitness, lo que podría proporcionar un notable aumento en la eficiencia de estas técnicas.

- También podría utilizarse el SOM para la creación de estrategias de inicialización de las poblaciones iniciales que emplean los algoritmos evolutivos, de forma que partiese de poblaciones lo más equilibradas posibles dentro del conjunto del espacio de búsqueda.

Apéndice A

Problemas y funciones de prueba

A.1. Función marea

Este problema consiste en buscar el máximo de la función marea, que matemáticamente se define como:

$$f(x,y) = 1 + \frac{sin(\sqrt{x^2+y^2})}{\sqrt{x^2+y^2}} \qquad x,y \in [-25,25]$$

Como podemos ver en la figura A.1, esta es una función bidimensional formada por anillos consecutivos de máximos y mínimos locales sobre el plano XY cuyo valor se va haciendo más próximo a 1 a medida que nos alejamos del único máximo global en el punto $< 0,0 >$ que alcanza el valor 2.

La forma más simple de representar las soluciones de este problema es un par de números reales. El valor de la solución puede representarse mediante un único número real.

Esta función ha sido utilizada como función de prueba en varios artículos relacionados con la CE tales como (Romero *et al.* , 2000).

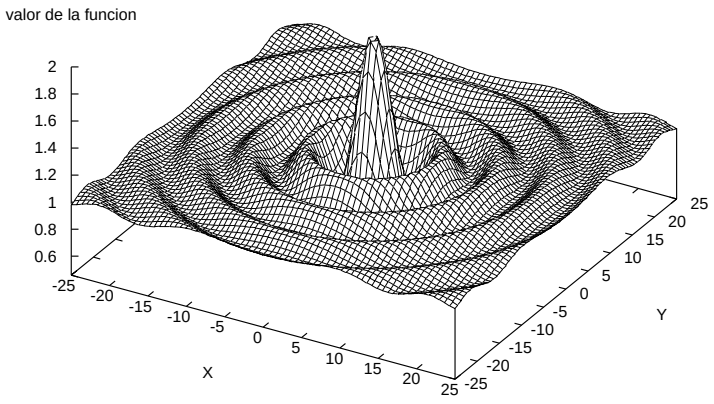


Figura A.1: Función marea: $1 + \frac{sin(\sqrt{x^2+y^2})}{\sqrt{x^2+y^2}}$

A.2. El problema de la mochila binaria

En el problema de la mochila binaria un conjunto de n objetos se deben introducir en una mochila de capacidad C. Cada objeto tiene un peso P_i y un beneficio B_i asociados, y el problema consiste en seleccionar un subconjunto de objetos cuyo peso total no exceda la capacidad de la mochila y cuyo beneficio sea máximo.

Hay una gran variedad de problemas de este tipo, muchos de los cuales son conocidos por ser NP-duros. Para nuestros experimentos hemos escogido un problema de la mochila binaria descrito en (Michalewicz, 1996) página. 81, en su variedad fuertemente correlacionada, con capacidad media y empleando un algoritmo de penalización lineal para calcular el fitness y asegurar la validez de los resultados. En (Martello *et al.* , 1997) podemos ver una recopilación de los métodos algorítmicos más importantes, y en (Hinterding, 1994) podemos encontrar varias formas de resolución mediante algoritmos evolutivos. A continuación describiremos brevemente el este problema:

Dados un conjunto de pesos P, otro de beneficios B y una capacidad C,

el problema consiste en encontrar un vector binario $x \in \mathbb{Z}_2^n$ tal que

$$\sum_{i=1}^{n} x_i \cdot P_i \leq C$$

y para el cual la suma de beneficios

$$\sum_{i=1}^{n} x_i \cdot B_i$$

sea máxima.

El conjunto de pesos, P, se ha escogido de forma aleatoria entre 1 y 10 y el de beneficios, B, se calcula a través de P mediante la fórmula

$$B_i = P_i + 5$$

La función de penalización lineal empleada ha sido:

$$pen(x) = \rho \cdot \left(\sum_{i=1}^{n} x_i \cdot P_i - C \right) \qquad \rho = max_{i=1..n}\{B_i/P_i\}$$

En nuestros experimentos utilizamos $n = 500$, con lo cual el espacio de búsqueda es el hipercubo $\mathbb{Z}_2^{500}$, una capacidad de mochila de $C = 1389$ y un valor de $\rho = 5$. El vector de pesos empleado es: <7, 1, 4, 6, 5, 6, 3, 3, 3, 4, 10, 6, 10, 6, 1, 10, 3, 5, 2, 8, 2, 4, 10, 1, 1, 9, 10, 8, 4, 6, 1, 9, 5, 1, 3, 4, 6, 10, 2, 10, 2, 8, 8, 4, 5, 9, 10, 6, 9, 6, 1, 3, 10, 3, 7, 8, 6, 2, 4, 1, 1, 6, 7, 6, 10, 1, 3, 1, 5, 1, 6, 10, 10, 10, 4, 4, 6, 5, 6, 5, 5, 8, 7, 6, 10, 7, 6, 10, 2, 7, 1, 9, 6, 5, 6, 7, 7, 2, 2, 1, 6, 7, 8, 8, 10, 9, 9, 2, 6, 3, 3, 10, 9, 1, 6, 2, 7, 2, 8, 4, 8, 6, 8, 6, 10, 2, 1, 3, 9, 8, 10, 7, 6, 8, 4, 5, 3, 3, 4, 1, 6, 3, 3, 2, 1, 8, 3, 5, 4, 8, 10, 4, 8, 8, 3, 5, 2, 1, 7, 7, 2, 2, 1, 1, 4, 2, 10, 4, 8, 2, 7, 2, 10, 2, 5, 9, 7, 9, 9, 1, 1, 3, 10, 8, 8, 6, 8, 10, 9, 5, 4, 10, 2, 4, 5, 4, 2, 5, 1, 10, 7, 6, 10, 1, 9, 10, 9, 8, 2, 9, 6, 7, 2, 7, 2, 7, 2, 6, 9, 10, 4, 8, 2, 5, 6, 4, 6, 9, 7, 2, 4, 4, 2, 7, 3, 3, 4, 4, 5, 6, 7, 2, 2, 7, 6, 4, 7, 1, 2, 2, 8, 10, 3, 4, 2, 3, 10, 6, 4, 10, 8, 4, 7, 9, 3, 9, 3, 8, 7, 7, 10, 4, 3, 4, 10, 6, 4, 1, 1, 3, 4, 10, 6, 3, 6, 1, 10, 6, 2, 4, 7, 8, 5, 3, 4, 9, 8, 6, 7, 3, 7, 8, 6, 7, 2, 8, 6, 5, 10, 6, 2, 8, 1, 4, 4, 8, 10, 3, 6, 9, 6, 9, 4, 9, 3, 10, 7, 7, 7, 10, 9, 2, 5, 6, 4, 5, 3, 8, 4, 7, 9, 9, 9, 5, 8, 3, 2, 7, 4, 1,

3, 1, 9, 10, 4, 10, 3, 3, 5, 5, 1, 7, 5, 6, 3, 5, 2, 9, 10, 7, 6, 7, 6, 8, 6, 9, 10, 7, 6, 9, 4, 4, 9, 1, 5, 9, 1, 9, 9, 6, 3, 2, 7, 4, 5, 4, 9, 6, 4, 5, 9, 6, 3, 5, 6, 7, 7, 2, 1, 2, 1, 10, 7, 10, 3, 4, 9, 7, 9, 9, 1, 5, 2, 4, 4, 1, 10, 6, 2, 9, 5, 8, 10, 3, 8, 6, 3, 4, 9, 9, 2, 2, 7, 5, 2, 10, 3, 7, 8, 10, 7, 1, 4, 6, 5, 10, 2, 10, 8, 10, 3, 6, 3, 4, 6, 3, 7, 4, 5, 3, 8, 8, 1, 5, 6, 8, 5, 6, 7, 5, 3, 8, 1, 8, 7, 1, 7, 3, 7, 7, 7, 9, 1, 9, 3, 8, 3, 3, 4, 10>.

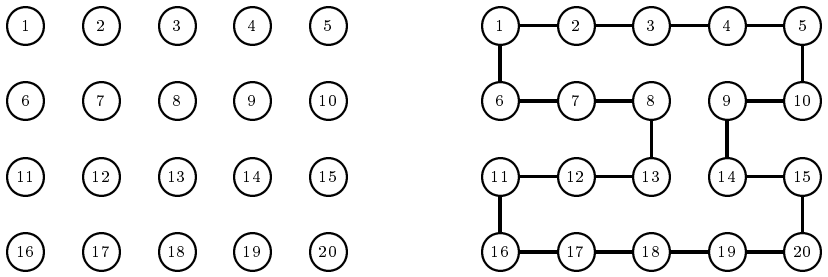
En esta instancia del problema, el valor del óptimo global es 3109 y a pesar de haber $\binom{55}{42} \cdot \binom{45}{2} \approx 1{,}44 \cdot 10^{15}$ diferentes combinaciones de objetos en la mochila que nos permiten conseguirlo no hemos conseguido obtenerlo en ninguna ejecución. El máximo valor que hemos obtenido es 3104 (ver figura 4.9). Esto se debe sin duda a la dificultad del problema, ante cuyo espacio de búsqueda de casi 2^{500} posibles combinaciones (salvo restricciones), el número de óptimos es insignificante.

A.3. El problema del viajante de comercio

Este problema consiste en recorrer un cierto número de ciudades sin pasar dos veces por la misma y volviendo al punto de partida inicial de forma que la distancia recorrida sea la mínima posible. Para una explicación más extensa consultar (Lawer *et al.* , 1985). El nombre del problema proviene del inglés "Travelling Salesman Problem" o TSP. Es un problema de optimización combinatoria que aparece en numerosas aplicaciones. Matemáticamente hablando, consiste en la búsqueda de un ciclo hamiltoniano mínimo en un grafo completo de n nodos. Se ha demostrado que este problema es NP-duro (Garey & Johnson, 1979).

Veamos el ejemplo que se muestra en la figura A.2. Supongamos que deseamos recorrer 20 ciudades repartidas en un plano y que existe un camino uniendo a cada una de ellas con todas las demás. A la izquierda mostramos las ciudades que deseamos recorrer. A la derecha una de las varias soluciones óptimas que tiene este caso concreto.

Existe varias formas de representar las soluciones de este problema. Podemos ver algunas de ellas en el capítulo 10 de (Michalewicz, 1996). La más intuitiva es la que crea una lista ordenada de principio a fin con las ciudades



(a) TSP con 20 ciudades (b) Posible solución

Figura A.2: Ejemplo del problema del viajante de comercio. A la izquierda, (a), podemos ver el conjunto de 20 ciudades que se deben recorrer. A la derecha, (b), una de las posibles soluciones óptimas para este problema, cuya longitud es 20.

que se deben recorrer. Mediante esta representación la solución mostrada en la figura A.2 se vería así:

(1 2 3 4 5 10 9 14 15 20 19 18 17 16 11 12 13 8 7 6)

La suma de las distancias recorridas entre ciudades es el valor de la solución que representada, que en este mismo ejemplo es 20.

Existen dos variantes de este problema. La que hemos visto es conocida como TSP, de "Travelling Salesman Problem", y en ella los caminos entre ciudades pueden recorrerse en ambas direcciones. La otra variante es conocida como ATSP, de "Asimetric Travelling Salesman Problem", y se diferencia de la anterior en que las soluciones no son simétricas, es decir, no ha de haber un camino de igual longitud para ir de una ciudad a otra que para volver.

A.4. Función F2 de Schaffer

Este problema consiste en buscar un valor real, x, tal que se minimicen simultáneamente los valores de las dos siguientes funciones:

$$f_{21}(x) = x^2 \qquad f_{22}(x) = (x-2)^2 \qquad x \in [-6, 6]$$

Este es un problema clásico de optimización multiobjetivo propuesto por

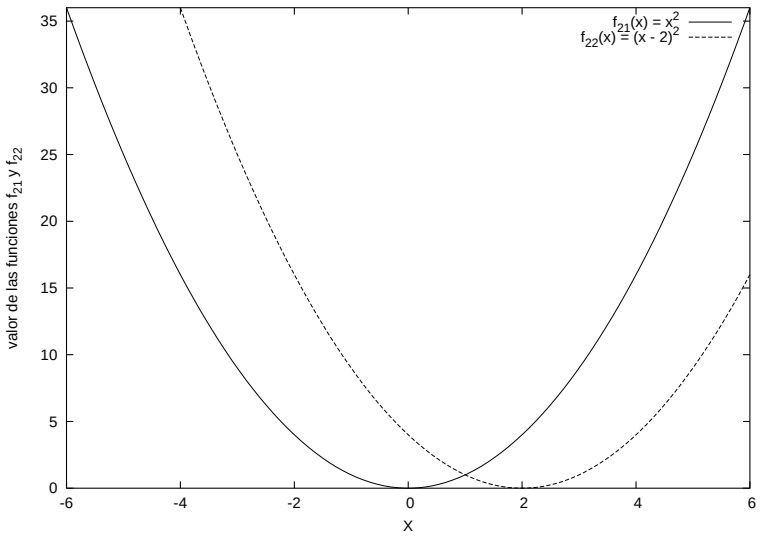


Figura A.3: Función F2 de Schaffer: minimizar simultáneamente las dos funciones $f_{21}(x) = x^2, f_{22}(x) = (x-2)^2 x \in [-6, 6]$

Schaffer en (Schaffer, 1984). Ha sido utilizado como ejemplo de prueba en multitud de trabajos que buscan un frente de pareto-óptimos o soluciones no dominadas. Se dice que una solución domina a otra si resuelve mejor alguno de los subobjetivos y en los demás es al menos igual de buena. En este caso concreto de la función F2, un solución **a** dominaría a otra **b** si y sólo si

$$\begin{aligned} f_{21}(a) < f_{21}(b) \quad & y \quad f_{22}(a) \leq f_{22}(b) \\ & o \\ f_{21}(a) \leq f_{21}(b) \quad & y \quad f_{22}(a) < f_{22}(b) \end{aligned}$$

En la figura A.4 podemos ver el resultado de la ejecución de un algoritmo evolutivo que busca un frente de pareto-óptimos para este problema. Las poblaciones inicial y final de un misma ejecución son dibujadas de dos formas distintas. En la primera fila se muestra cada individuo sobre una las funciones f_{21} y f_{22}. En la segunda se usan esos dos valores de cada individuo como coordenadas X e Y para dibujarlos sobre un plano.

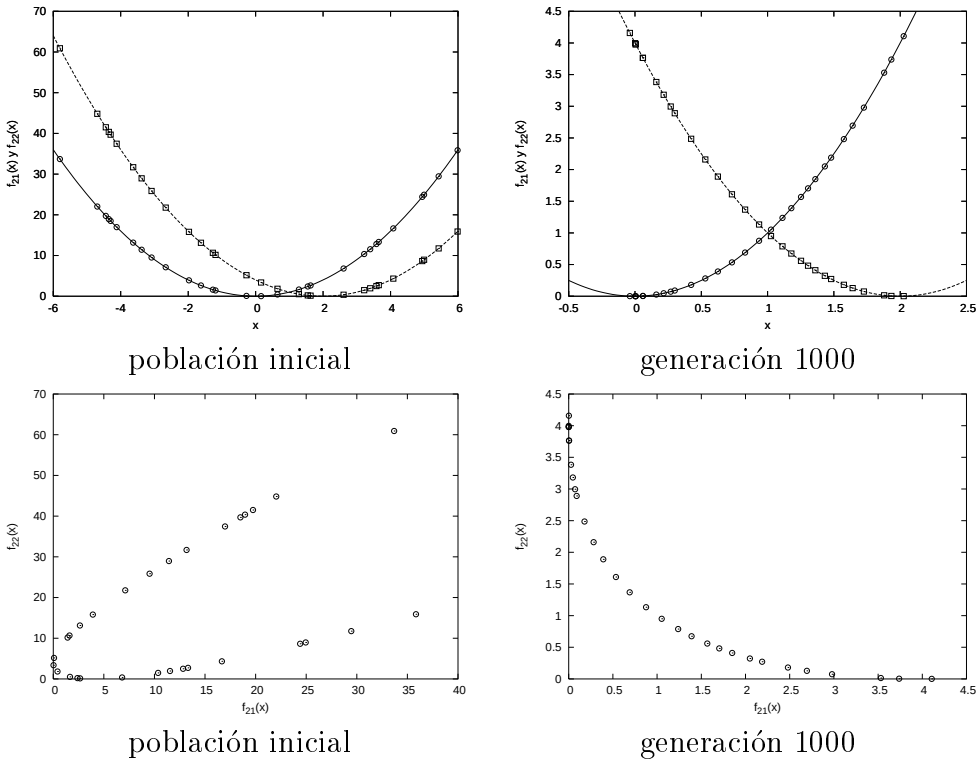


Figura A.4: Genotipo de la función F2 de Schaffer. Dos formas distintas de ver las poblaciones inicial y final de una ejecución de un algoritmo evolutivo que resuelve este problema. En la primera fila se muestra cada individuo sobre una las funciones f_{21} y f_{22}. En la segunda se usan esos dos valores como coordenadas X e Y para dibujarlos en el plano.

Apéndice B

Bibliografía

Bibliografía

Arthur, W. Brian. 1990. Positive feedbacks in the economy. *Scientific American*, **262**(Feb), 92–99.

Bäck, T. 1996. Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, and Genetic Algorithms. *Oxford University Press, New York.*

Bäck, T., Hoffmeister, F., & Schwefel, H.P. 1991. A survey of evolution strategies. *In R. K. Belew and L.B. Booker, eds., Proceedings of the Fourth International Conference on Genetic Algorithms. Morgan Kaufmann.*

Bedau, Mark A., & Brown, C. Titus. 1999. Visualizing Evolutionary Activity of Genotypes. *Artificial Life*, **5**(1), 17–35.

Bedau, Mark A., Joshi, S., & Lillie, B. 1999. Visualizing Waves of Evolutionary Activity of Alleles. *In:* (Wu, 1999).

Blake, C.L., & Merz, C.J. 1998. *UCI Repository of machine learning databases.* University of California, Irvine, Dept. of Information and Computer Sciences. `http://www.ics.uci.edu/ mlearn/MLRepository.html`.

Booch, G. 1994. *Object-Oriented Analysis and Design, with Applications.* sedond edn. San Mateo, California: Benjamin/Cummings.

Box, G.E.P. 1957. Evolutionary operation: A method for increasing industrial productivity. *Journal of the Royal Statistical Society C 6, no. 2: 81-101.*

Bremermann, H.J. 1962. Optimization through evolution and recombination. *In M.C. Yovits, G.T. Jacobi, and G.D. Goldstein, eds., Self-Organizing systems. Spartan Books.*

Castillo, P.A. 2000. *Optimización de Perceptrones Multicapa Mediante Algoritmos Evolutivos.* Ph.D. thesis, Universidad de Granada.

Castillo, P.A., Carpio, J., Merelo, J. J., Rivas, V., Romero, G., & Prieto, A. 2000a. Evolving Multilayer Perceptrons. *Neural Processing Letters*, **12**(2), 115–127.

Castillo, P.A., Merelo, J. J., Rivas, V., Romero, G., & Prieto, A. 2000b. G-Prop: Global Optimization of Multilayer Perceptrons using GAs. *Neurocomputing, Vol.35/1-4*, 149–163.

Castillo, P.A., Merelo, J.J., Prieto, A., Rojas, I., & Romero, G. 2002. Statistical analysis of the parameters of a neuro-genetic algorithm. *IEEE Transactions on Neural Networks*, **13**(6), 1374–1394.

Coello, Carlos A. 2000. An updated survey of GA-based multiobjective optimization techniques. *ACM Computing Surveys*, **32**(2), 109–143.

Collins, John James. 1999. Visualization of evolutionary algorithms using principal component analysis. *In:* (Wu, 1999).

Collins, Trevor D. 1996. *Genotypic-Space Mapping: Population Visualization for Genetic Algorithms.* The Knowledge Media Institute, The Open University, Milton Keynes, UK, Technical Report KMI-TR-39, 30th September 1996.

Collins, Trevor D. 1997. Using software visualization technology to help evolutionary algorithm users validate their solutions. *Pages 307–314 of: Proceedings of the Seventh International Conference on Genetic Algorithms ICGA'97.*

Collins, Trevor D. 1998a. *The Application of Software Visualization Technology to Evolutionary Computation: A case study in genetic algorithms.* Ph.D. thesis, The Open University, Knowledge Media Institute, Milton Keynes, UK.

Collins, Trevor D. 1998b. Understanding evolutionary computing: A hands on approach. *Pages 564–569 of: Proceedings of the IEEE International Conference on Evolutionary Computation ICEC'98.*

Corne, D., Ross, P., & Fang, H.L. 1994. *Fast practical evolutionary timetabling.* Lecture Notes in Computer Science, vol. 865. Berlin, German: Springer-Verlag. Pages 251–263.

Cox, T.F., & Cox, M.A.A. 1994. *Multidimensional Scaling.* London: Chapman & Hall.

Dabs, T., & Schoof, J. 1995. *GIGA - A Graphical User Interface for Genetics Algorithms.* Tech. rept. 98. Lehrstuhl Informatik II, Univ. Würzburg, Würzburg, Germany.

Darwin, Charles. 1859. *On the Origin of Species by Means of Natural Selection: or the Preservation of Favoured Races in the Struggle for Life.* London: John Murray.

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

Demartines, Pierre, & Hérault, Jeanny. 1997. Curvilinear component analysis: a self organizing neural network for non linear mapping of data sets. *IEEE Transactions on Neural Networks*, **8**, 148–154.

Dolan, Ariel. 1998. GA-Playground. Disponible en Internet en la dirección: `http://www.aridolan.com/ga/gaa/gaa.html`.

ECVW. 1999. *Annie S. Wu, Ed., Proceedings of the Genetic and Evolutionary Computation Conference, Evolutionary Computation Visualization Workshop. 1999.* San Mateo, CA: Morgan Kaufmann.

EIBEN, A.E., VAN KEMENADE, C.H.M., & KOK, J.N. 1995. Orgy in the Computer: Multi-Parent Reproduction in Genetic Algorithms. *in Proc. of The Third European Conference on Artificial Life (ECAL'95). Lecture Notes in Artificial Intelligence, vol. 929, pp.935-945. F. Morán, A. Moreno, J.J. Merelo, P. Chacón (Eds.). Springer-Verlag. Granada, Spain.*

FAHLMAN, S.E. 1988. Faster-Learning Variations on Back-Propagation: An Empirical Study. *In: Proceedings of the 1988 Connectionist Models Summer School.* Morgan Kaufmann.

FLEXER, ARTHUR. 1997. Limitations of self-organizing maps for vector quantization and multidimensional scaling. *Pages 445–51 of:* MOZER, M. C., JORDAN, M. I., & PETSCHE, T. (eds), *Advances in Neural Information Processing Systems 9. Proceedings of the 1996 Conference.* London, UK: MIT Press.

FLEXER, ARTHUR. 1999. On the Use of Self-Organizing Maps for Clustering and Visualization. *Pages 80–88 of: Principles of Data Mining and Knowledge Discovery.*

FOGEL, D.B. 1995. Evolutionary Computation: Toward a New Philosophy of Machine Intelligence. *IEEE Press.*

GAREY, M., & JOHNSON, D. 1979. *Computers and Intractability.* San Francisco: W.H. Freeman.

GARNIER, JOSSELIN, KALLEL, LEILA, & SCHOENAUER, MARC. 1999. Rigorous Hitting Times for Binary Mutations. *Evolutionary Computation*, **7**(2), 173–203.

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

GOLDBERG, D.E., & RICHARDSON, J. 1987. Genetics Algorithm with Sharing for Multimodal Function Optimization. *Pages 41–49 of:* GREFENSTETTE, J.J. (ed), *Proceedings of the First International Conference on Genetic Algorithms.* Hillsdale, NJ: Lawrence Erlbaum Associates.

GORMAN, R.P., & SEJNOWSKI, T.J. 1988. Analysis of Hidden Units in a Layered Network Trained to Classify Sonar Targets. *Neural Networks*, **1**, 75–89.

GRÖNROOS, M.A. 1998. Evolutionary Design of Neural Networks. *Master of Science Thesis in Computer Science. Department of Mathematical Sciences. University of Turku.*

HARIK, G. R., LOBO, F. G., & GOLDBERG, D. E. 1999. The Compact Genetic Algorithm. *IEEE - Evolutionary Computation*, **3**(4), 287.

HART, EMMA, & ROSS, PETER. 2001. GAVEL - A New Tool for Gentic ALgorithm Visualization. *IEEE Transactions on Evolutionary Computation*, **5**(4), 335–348.

HENNING, M., & VINOSKI, S. 1999. Advanced CORBA Programming with C++. *Addison-Wesley Professional Computing Series. ISBN 0-201-37927-9. Addison-Wesley Longman, Inc.*

HEYLIGHEN, F. 1992. Selfish memes and the evolution of cooperation. *Journal of Ideas, 2(4):77-84.*

HINTERDING, ROBERT. 1994. Mapping Order-independent Genes and the Knapsack Problem. *Pages 13–17 of: Proc. of the First IEEE Int. Conf. on Evolutionary Computation.* IEEE Press.

HOLLAND, J.H. 1975. Adaptation in Natural and Artificial Systems. *University of Michigan Press (Second Edition: MIT Press, 1992).*

HOTELLING, H. 1933. Analysis of a complex of statistical variables into principal components. *Journal of Educational Psychology*, **24**, 417–441,498–520.

IGEL, CHRISTIAN, & KREUTZ, MARTIN. 1999. Using Fitness Distributions to Improve the Evolution of Learning Structures. *Pages 1902–1909 of:* ANGELINE, PETER J., MICHALEWICZ, ZBYSZEK, SCHOENAUER, MARC,

YAO, XIN, & ZALZALA, ALI (eds), *Proceedings of the Congress on Evolutionary Computation*, vol. 3. Mayflower Hotel, Washington D.C., USA: IEEE Press.

JAKOB, W., GEORGES-SCHELEUTER, M., & BLUME, C. 1992. Application of genetic algorithms to task planning and learning. *Pages 291–300 of:* MÄNNER, R., & MANDERICK, B. (eds), *Proceedings of the Parallel Problem Solving from Nature - PPSN'1992*. Lecture Notes in Computer Science. Amsterdam: North-Holland Publishing Company.

JONES, T. 1995. Crossover, macromutation and population based search. *Pages 73–80 of:* ESHELMAN, L. (ed), *Proceedings of the 6th International Conference on Genetic Algorithms*.

KASKI, SAMUEL. 1997. Data Exploration Using Self-Organizing Maps. *Acta Polytechnica Scandinavica, Mathematics, Computing and Management in Engineering Series No. 82*, March.

KEIJZER, M., MERELO, J. J., ROMERO, G., , & SCHOENAUER, M. 2001. Evolving Objects: a general purpose evolutionary computation library. *Pages 231–244 of: Proceedings Evolution Artificielle EA'2001*. Lecture Notes in Computer Science, no. 2310. Springer-Verlag.

KIRKPATRICK, S. 1984. Optimization by Simulated Annealing - Quantitative Studies. *J. Stat. Phys. 34, 975-986.*

KOHONEN, TEUVO. 1990. The Self-Organizing Map. *Pages 1464–1480 of: Proceedings of the IEEE*, vol. 78.

KOHONEN, TEUVO. 1997. *The Self-Organizing Maps*. second extended edn. Information Sciences, vol. 30. Springer-Verlag.

KOHONEN, TEUVO. 1998. The self-organizing map. *Neurocomputing*, **21**, 1–6.

KOHONEN, TEUVO, HYNNINEN, JUSSI, KANGAS, JARI, & LAAKSONEN, JORMA. 1996. *SOM_PAK: The Self-Organizing Map Program Package.*

Tech. rept. A31. Helsinki University of Technology, Laboratory of Computer and Information Science, FIN-02150 Espoo, Finland.

KÖNIG, ANDREAS. 1998 (October). A Survey of Methods for Multivariate Data Projection, Visualisation and Interactive Analysis. *Pages 55–59 of: Proceedings of the 5th International Conference on Soft Computing and Information/Intelligent Systems (IIZUKA'98).*

KORST, E.H.L. AARTS; J. 1989. Simulated Annealing and Boltzmann Machines. *John Wiley, Chechester, U.K.*

KOZA, JOHN R. 1992. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* Cambridge, MA: The MIT Press.

KRUSKAL, J. B. 1964. Multidimensional scaling by optimizing goodness to fit to a nonmetric hypothesis. *Psychometrika*, **29**, 1–27.

LAWER, L.E., LENSTRA, J.K., KAN, A.H.G RINOOY, & SHMOYS, D.B. 1985. *The Travelling Salesman Problem.* Chichester, UK: John Wiley.

LEE, JOHN ALDO, LENDASSE, AMAURY, DONCKERS, NICOLAS, & VERLEYSEN, MICHAEL. 2000 (April). A robust nonlinear projection method. *Pages 13–20 of: ESANN 2000, 8th European Symposium on Artificial Neural Networks.*

LEE, JOHN ALDO, LENDASSE, AMAURY, & VERLEYSEN, MICHAEL. 2002 (April). Curvilinear Distance Analysis versus Isomap. *Pages 185–192 of: ESANN 2002, 10th European Symposium on Artificial Neural Networks.*

LENDASSE, A., LEE, JOHN ALDO, WERTZ, V., , & VERLEYSEN, MICHAEL. 2000 (April). Time Series Forecasting using CCA and Kohonen Maps - Application to Electricity Consumption. *Pages 329–334 of: ESANN 2000, 8th European Symposium on Artificial Neural Networks.*

LINDE, ANDREI. 1994. The self-reproducing inflationary universe. *Scientific American, 271(5):48-55. Available also from* `http://www.sciam.com/-specialissues/0398cosmos/0398linde.html`.

MANGASARIAN, O. L., SETIONO, R., & WOLBERG, W.H. 1990. Pattern recognition via linear programming: Theory and application to medical diagnosis. *Large-scale numerical optimization*, 22–30.

MARTELLO, S., PISINGER, D., & TOTH, P. 1997. New trends in exact algorithms for the 0-1 knapsack problem. *Pages 151–160 of:* BARCELÓ, J. (ed), *Proceedings of EURO/INFORMS-97.*

MAYNARD-SMITH, J. 1997. *The theory of evolution.* 3 edn. Cambridge University Press.

MERELO, J.J., CARPIO, J., CASTILLO, P.A., RIVAS, V.M., & ROMERO, G. 1999. Evolving Objects. *Technical report available at* `http://-geneura.ugr.es/ jmerelo/EOPaper`.

MERELO, J.J., CARPIO, J., CASTILLO, P.A., RIVAS, V., & ROMERO, G. 2000. Evolving Objects. *in Proc. of Int'l Workshop on Evolutionary Computation (IWEC'2000) pp. 202-208. State Key Laboratory of Software Engineering. Wuhan University.*

MICHALEWICZ, ZBIGNIEW. 1996. *Genetic Algorithm + Data Structure = Evolution Programs, Third, Extended Edition.* Springer-Verlag.

MICROSOFT. 2000. Microsoft COM Technologies - Information and Resources for the Component Object Model-based technologies. *Página web en* `http://www.microsoft.com/com`.

MODHA, D.S., SPANGLER, S., & VAITHYANATHAN, S. 1998. *Multidimensional cluster visualization using guided tours.* Technical Report Reseach Report RJ 10124, IBM Almaden Research Center, San Jose, CA, June 17 1998.

OMG. 2000. Object Management Group Home Page. *Website at* `http://-www.omg.org`.

OSYCZKA, A. 1985. *Multicriteria otimization for engineering design.* Design Optimization. Academic Press. Pages 193–227.

PERONA, P., & POLITO, M. 2002. Grouping and dimensionality reduction by locally linear embedding. *In:* DIETTERICH, T. G., BECKER, S., & GHAHRAMANI, Z. (eds), *Advances in Neural Information Processing Systems 14.* Cambridge, MA: MIT Press.

POHLHEIM, HARTMUT. 1999. Visualization of Evolutionary Algorithms - Set of Standard Techniques and Multidimensional Visualization. *In:* (Wu, 1999).

PRECHELT, LUTZ. 1994 (September). *PROBEN1 - A Set of Benchmarks and Benchmarking Rules for Neural Network Training Algorithms.* Tech. rept. 21/94. Fakultät für Informatik, Universität Karlsruhe, D-76128 Karlsruhe, Germany.

RADCLIFFE, N.J. 1991. Equivalence class analysis of genetic algorithms. *Complex Systems 5, no.2: 183-205.*

RAY, T.S. 1992. An evolutionary approach to the synthesis of life. *In C.G. Langton, C. Tylor, D. Farmer ans S. Rasmussen, (Eds.), Artificial Life II, Santa Fe Institute Studies in the Sciences of Complexity*, 371–408.

RECHENBERG, I. 1965. Cybernetic solution path of an experimental problem. *Ministry of Aviation, Royal Aircraft Establishment (U.K.).*

RECHENBERG, I. 1973. Evolutionsstrategie: optimierung Technischer Systeme nach Prinzipien der Biologischen Evolution. *Frommann-Hozboog (Stuttgart).*

REED, RUSSEL D., & MARKS, ROBERT J. 1999. *Neural Smithing: supervised learning in feedforward artificial neural networks.* The MIT Press.

RIOLO, R.L. 1987. Bucket Brigade performance: Il Default Hierchies. *Pages 196–201 of: Second International Conference on Genetic Algorithms and their Applications.*

RIPLEY, B.D. 1996. *Pattern Recognition and Neural Networks.* Cambridge, GB: Cambridge University Press.

Rojas, I., González, J., Pomares, H., Merelo, J.J., Castillo, P.A., & Romero, G. 2002. Statistical Analysis of the Main Parameters Involved in the Design of a Genetic ALgorithm. *IEEE Transactions on Systems, Man and Cybernetics*, **32**(1), 31–37.

Romero, G., Arenas, M. G., Castellano, J. G., Castillo, P.A., Carpio, J., Merelo, J. J., Prieto, A., & Rivas, V. 2000. Evolutionary Computation Visualization: Application to G-PROP. *Pages 902–912 of:* Schoenauer, Marc, Deb, Kalyanmoy, Rudolph, Günter, Yao, Xin, Lutton, Evelyne, Merelo, J.J., & Schwefel, Hans-Paul (eds), *Proceedings of the 6th Parallel Problem Solving from Nature - PPSN VI.* Lecture Notes in Computer Science, no. 1917. Springer-Verlag.

Romero, G., Castillo, P.A., Merelo, J. J., & Prieto, A. 2001. Using SOM for Neural Network Visualization. *Pages 629–636 of:* Mira, José, & Prieto, Alberto (eds), *Proceedings of the 6th International Workshop on Artificial Neural Networks - IWANN 2001.* Lecture Notes in Computer Science, no. 2084. Granada. Spain: Springer-Verlag.

Romero, G., Merelo, J. J., Castillo, P.A., Castellano, J. G., & Arenas, M. G. 2002. Genetic Algorithm Visualization using Self-Organizing Maps. *Pages 442–451 of:* Merelo Guervós, J.J., Adamidis, Panagiotis, Beyer, Hans. Georg, Fernandez Villacañas, José Luis, & Schwefel, Hans-Paul (eds), *Proceedings of the 7th Parallel Problem Solving from Nature - PPSN VII.* Lecture Notes in Computer Science, no. 2439. Springer-Verlag.

Roweis, Sam T., & Saul, Lawrence K. 2000. Nonlinear Dimensionality Reduction by Locally Linear Embedding. *Science*, **290**(December), 2323–2336.

Sammon Jr., J.W. 1969. A nonlinear mapping for data structure analysis. *IEEE Transactions on Computers*, **18**, 401–409.

SCHAFFER, J.D. 1984. *Some experiments in machine learning using vector evaluated genetic algorithm.* Ph.D. thesis, Vanderbilt University.

SCHAFFER, J.D. 1985. Multiple objetive optimization with vector evaluated genetic algorithms. *Pages 93–100 of: Proceedings of the First International Conference on Genetic Algorithms - ICGA'1985.* London: Lawrence Erlbaum.

SCHOENAUER, J.J. MERELO; M. G. ARENAS; J. CARPIO; P.A. CASTILLO; V. M. RIVAS; G. ROMERO; M. 2000. Evolving Objects. *In M. Graña, editor, FEA (Frontiers of Evolutionary Algorithms) proceedings*, 1083–1086. ISBN: 0-9643456-9-2.

SCHWEFEL, H.P. 1975. Evolutionsstrategie und numerische optimierung. *Ph.D. thesis, Technische Universitat Berlin.*

SCHWEFEL, H.P. 1977. Numerische optimierung von Computer-Modellen mittels der Evolutionsstrategie. *Basel: Birkhauser.*

SCHWEFEL, H.P. 1995. Evolution and Optimum Seeking. *Wiley, New York.*

SHEPARD, R.N. 1962. The analysis of proximities: multidimensional scaling with an unknown distance function. *Psychometrika*, **27**, 125–140,219–246.

SHINE, W., & EICK, C. 1997. Visualizing the evolution of genetic algorithm search process. *Pages 367–372 of: Proceedings of the 1997 IEEE International Conference on Evolutionary Computation.* Piscataway, NJ: IEEE Press.

SYSWERDA, G., & PALMUCCI, J. 1991. The Application of Genetic Algorithms to Resource Scheduling. *Pages 502–508 of:* BELEW, R.K., & BOOKER, L.B. (eds), *Proceedings of the Fourth International Conference on Genetic Algorithms - ICGA'1991.* San Mateo, California: Morgan Kaufmann.

Teller, N. Metropolis; A.W. Rosenbluth; M.N. Rosenbluth; A.H. Teller; E. 1958. Equations of State Calculations by Fast Computing Machines. *J. Chem. Phys. 21,1087-1092.*

The MathWorks, Inc. 1994. *MATLAB.* `http://www.mathworks.com`.

Toombs, J. Reed; R., & Barricelli, N. A. 1967. Simulation of biological evolution and machine learning. *Journal of Theoretical Biology 17: 319-342.*

Toussaint, Marc, & Igel, Christian. 2002. Neutrality: A Necessity for Self-Adaptation. *Pages 1354–1359 of: Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2002).*

Trogerson, W. S. 1952. Multidimensional scaling: I. Theory and method. *Psychometrika*, **17**, 401–419.

Tsogo, L., & Masson, M. 2000. Multidimensional Scaling methods for many-objects sets: a review. *Multivariate Behavioral Research*, **35**(3), 307–320.

Tuson, A., & Ross, P. 1998. Adapting operator settings in genetic algorithms. *Evolutionary Computation*, **6**(2), 161–184.

Ultsch, A. 1993. Self organized feature maps for monitoring and knowledge adquisition of a chemical process. *Pages 864–867 of:* S. Gielen, B. Kappen (ed), *International Conference on Artificial Neural Networks (ICANN'93).* London: Springer-Verlag.

Vecchi, S. Kirkpatrick; C.D. Gerlatt; M.P. 1983. Optimization by Simulated Annealing. *Science 220, 671-680.*

Vesanto, Juha. 1999. SOM-Based Data Visualization Methods. *Intelligent Data Analysis*, **3**(2), 111–126.

Walsh, L.J. Fogel; A.J. Owens; M.J. 1966. Artificial Intelligence through Simulated Evolution. *Wiley.*

Whitley, L.D. 1989. The GENITOR algorithm and selection pressure: Why rank-based allocation of reproductive trials is best. *Pages 116–121 of:* Schaffer, J.D. (ed), *Proceedings of the Third International Conference on Genetic Algorithms*. Morgan Kaufmann.

Wu, Annie S. (ed). 1999. *Proceedings of the 1999 Genetic and Evolutionary Computation Conference*. Orlando, Florida, USA: Morgan Kaufmann.

Wu, Annie S., Ramsey, C., Jong, K. De, Grefenstette, J., & Burke, D. 1999a. VIS: A genetic algorithm visualization tool. *Pages 106–109 of:* Wu, Annie S. (ed), *Proceedings of the 1999 Genetic and Evolutionary Computation Conference*. Orlando, Florida, USA: Morgan Kaufmann.

Wu, Annie S., De Jong, Kenneth A., Burke, Donald S., Grefenstette, John J., & Ramsey, Connie Loggia. 1999b. Visual Analysis of Evolutionary Algorithms. *Pages 1419–1425 of:* Angeline, Peter J., Michalewicz, Zbyszek, Schoenauer, Marc, Yao, Xin, & Zalzala, Ali (eds), *Proceedings of the Congress on Evolutionary Computation (CEC'99)*, vol. 2. Mayflower Hotel, Washington D.C., USA: IEEE Press.

Yao, Xin. 1992. *A Review of Evolutionary Artificial Neural Networks*. Victoria, Australia: Commonwealth Scientific and Industrial Research Organization.

Apéndice C

Índice alfabético

Índice alfabético