UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

INGENERÍA TÉCNICA

EN INFORMÁTICA DE GESTIÓN

PROYECTO FIN DE CARRERA

Librería para Algoritmos Genéticos Basados en Nichos

Tutor: Inés María Galván León
Ricardo Aler Mur

Autor: Francisco Javier Florido Florido

Capítulo 1. Introducción

Sobre los años setenta, y gracias a los estudios realizados por Holland surgió una de las líneas más prometedoras de la inteligencia artificial, la de los algoritmos genéticos (AG´s) Es una técnica de programación que imita a la evolución biológica para resolución de problemas. El AG de Holland consiste en evolucionar una población de individuos representadas por cromosomas a un conjunto de soluciones potenciales. Para obtener estos resultados se evalúa cada candidato con la función, cuyo nombre es función de fitness que decide qué individuos se han adaptado mejor (son los que sobreviven), y cuáles son menos aptos, los cuales son descartados y eliminados de la población. Los individuos prometedores se conservan y se les realizan modificaciones, obteniendo individuos que mejoraron y otros que han empeorado con los cambios en su código, siendo eliminados de nuevo los peores. Este proceso se repetirá un número determinados de veces, teniendo una gran posibilidad de localizar una solución al problema o una buena aproximación.

Aunque los AG´s simple han demostrado ser una estrategia poderosa y exitosa para resolver problemas, no están diseñados para encontrar todos los óptimos globales, convergiendo a una única solución, eliminado también los óptimos locales al ser descartados en la función optima al ganar en la comparación. Existen sin embargo los algoritmos de nichos, que extienden los AG, que localizan y mantienen las múltiples soluciones con los óptimos locales y globales.

Los AG´s de nicho permiten encontrar múltiples soluciones de un problema multimodal, ya que tiene capacidad de crear y mantener varias sub-poblaciones dentro de un determinado espacio de búsqueda. Cada una de esta sub-poblaciones corresponde a cada óptimo que se pretende encontrar de una determinada función multimodal.

El estudio que se va realizar cosiste en explicar las diferentes técnicas evolutivas, poniendo un hincapié en los AG´s y los AG´s de nichos. Además se ha implementado una librería de AG´s basados en nichos para localizar óptimos locales y globales. La eficacia del algoritmo será demostrado en varias funciones multimodales.

La memoria se va organizar en diferentes capítulos, los cuales se describen a continuación:

En el capítulo 2 se detallan las características generales de las Técnicas Evolutivas, indicando las diferentes técnicas que existen.

En el capítulo 3 se explicará con detalle los AG´s, su funcionamiento, sus componentes, comparándolos con otros métodos, para finalizar el capítulo con un pequeño resumen en aplicaciones y utilidades donde se están utilizando.

En el capítulo 4 se estudiarán los Algoritmos Genéticos de nichos para solucionar problemas multimodales. Se indicarán las diferentes técnicas que existen con sus ventajas e inconvenientes

En el capítulo 5 se explica la librería escrita para utilización de algoritmos genéticos con nichos en la resolución de problemas multimodales, detallando y explicando las principales funciones que se han realizado.

En el capítulo 6 se mostrará con ejemplos como funciona la librería que se ha creado y también se presentarán los resultados experimentales realizados para comprobar cómo el algoritmo obtiene los máximos de ciertas ecuaciones.

En el capítulo 7 se darán las conclusiones finales sobre el trabajo y también se indicarán algunas ideas para trabajos futuros.

Para terminar, en el capítulo 8 se presentará un presupuesto del proyecto con los costes de la realización del proyecto.

Capítulo 2. Técnicas Evolutivas

2.1 Características generales

Las técnicas evolutivas son utilizadas en multitud de campos con el fin de poder dar una posible inteligencia a las máquinas. Estas técnicas consisten en la creación de métodos que tratan de optimizar una función, utilizando una estrategia esencialmente independiente del problema que se trate, (Winston 1992). Estos métodos requieren muy poco conocimiento sobre el espacio de búsqueda o del problema en particular. La capacidad de adaptarse o de aprender, es una cualidad que suele tomarse como indicativo o referencia de la existencia de inteligencia. Las computadoras son excelentes en la realización de tareas repetitivas con precisión y velocidad. La adaptación y la búsqueda constituyen la capacidad de aprendizaje.

Las técnicas evolutivas son estrategias de búsqueda basadas en una población de individuos. El algoritmo se encuentra con una población que evoluciona con el fin de encontrar las mejores soluciones con respecto a una función dada, conocida como función de fitness. La población de individuos es sometida a diversas leyes de selección y alteración o mutación, de manera que los primeros individuos darán lugar a la creación de nuevos individuos cuyas características procederán parte de sus antecesores y parte de una forma aleatoria. Y estos nuevos individuos serán de nuevo sometidos a dicho proceso evolutivo, y así sucesivamente. De esta forma los individuos que supongan las mejores soluciones al problema, poseen mayor probabilidad de ser seleccionado e ir sobreviviendo a lo largo de la evolución

Para poder aplicar una técnica evolutiva es necesario:

- Una población de posibles soluciones debidamente representadas a través de individuos.

- Un procedimiento de selección basado en la aptitud de los individuos.

- Un procedimiento de transformación o construcción de nuevas soluciones a partir de las disponibles actualmente.

La relación de estos procedimientos con las ideas Darwinianas de la Selección Natural y la Evolución de las Especies es evidente. En realidad estos métodos tratan de emular el simple, cruel, incomparablemente y eficaz mecanismo por el cual las especies naturales han ido perfeccionándose y adecuándose a su entorno a lo largo de su historia. En una población de individuos, si uno de ellos presenta una pequeña ventaja sobre los demás producirá mejoras en las expectativas reproductivas, y con el transcurso de varias generaciones, se reproducirá en la totalidad de los individuos.

En el intento de emular procesos naturales, numerosos autores han aplicado técnicas de Evolución Natural de la Especies a las Técnicas Evolutivas, como Hillis 1990, Paradis 1995 y Juille 1993. El efecto final de dichos estudios es la obtención de soluciones más generales e independientes de la definición del entorno. La historia de la

Computación Evolutiva comienza con los Algoritmos Genéticos en las décadas de los 70 con Holland.

La función de optimización es la supervivencia de las especies. Si la Evolución Natural ha logrado individuos, especies y genes adecuados a entornos variadísimos, cambiantes y a menudo hostiles, esto es producto de un enorme grado de sofisticación y perfección ante problemas de semejante magnitud.

También debe destacarse el mecanismo de aleatoriedad, incluida en la Evolución Natural de las Especies. Si la generación de varios individuos nuevos a partir de dos padres siempre diera lugar a hijos idénticos, se mermaría la capacidad de exploración al reducir el número de observaciones diferentes en el espacio de búsqueda. Al incluir aleatoriedad en el cruce se asegura que los hijos tengan gran probabilidad de ser diferentes, eliminado la contrariedad anterior.

Por otro lado, el operador mutación constituye en esencia un "difusor" de movimientos aleatorios entre los miembros de una población, también con el objetivo de "saltar" a posiciones nuevas en la búsqueda, complementando al operador de cruce.

Generalmente también se dota de aleatoriedad a los procesos de selección involucrados en las técnicas evolutivas. El argumento es similar: permitiendo que los individuos peor dotados tengan alguna probabilidad de ser seleccionados se conseguirá que la búsqueda tenga un carácter más global y flexible posibilitando que alguno de los caminos de exploración sea en algunos momentos, poco propicio.

Los algoritmos evolutivos son suficientemente complejos para proveer un mecanismo de búsqueda lo suficientemente robusto y poderoso. Aunque no puede afirmarse que no realice una búsqueda aleatoria de una solución para un problema, el resultado es claramente no aleatorio.

En un método evolutivo genérico, una vez generado una población inicial P(0), se la someterá repetidas veces a los procesos de evaluación selección y procreación hasta que alguna condición de finalización sea satisfactoria.

2.2 Diferentes técnicas

Las ideas anteriormente expuestas pueden dar solución a problemas que se plantean en la Ingeniería o en otras ramas del saber y que al hombre le resultan complicados de resolver. Este argumento, al lado de las capacidades computacionales cada vez mayores de los ordenadores, parece haber desencadenado en las tres últimas décadas en el surgimiento de técnicas evolutivas, que resuelven un sinfín de problemas de optimización. Dentro de las técnicas evolutivas se encuentran: Algoritmos Genéticos, Estrategias Evolutiva, Programación evolutiva, Programación genética y Sistemas Clasificadores.

Las Estrategias Evolutivas codifican los genotipos como cadenas de números reales, y se usan para resolver problemas de optimización con un alto grado de dificultad. Su característica principal es la auto-adaptación de los parámetros de la estrategia, como por ejemplo la desviación típica de la amplitud de mutación. Fueron introducidas originalmente por Rechemberg 1973 y Schwefel 1984 para resolver

problemas de optimización continua imitando principios de la evolución natural, asociando el concepto de individuo o miembro a una solución factible del problema y el de la población a un conjunto de individuos (soluciones factibles).

La Programación Evolutiva evoluciona por mutación y selección una población de máquinas de estados finitos, con el objetivo de predecir los símbolos siguientes a partir de unos símbolos de entrada, lo que se emplea como un modelo de la inteligencia.

La Programación Genética es una metodología basada en el algoritmo evolutivo e inspirado en la evolución biológica para desarrollar automáticamente Es una técnica de aprendizaje automático utilizada para optimizar una población de programas o autómatas de acuerdo a una función llamada fitness que evalúa la capacidad.

Los sistemas clasificadores con aprendizaje son sistemas dinámicos, basados en reglas de producción, que aprenden por inducción a partir de ejemplos. Las reglas tienden al siguiente formato: *si <condición> entonces <acción>*, de forma que la acción se lleva a cabo si la condición es cierta. Estos sistemas se pueden considerar que son sistemas de producción que emplean los algoritmos evolutivos para generar las reglas. Pueden ser considerados híbridos entre sistemas de aprendizaje y sistemas evolutivos, con la salvedad de que no se utilizan para resolver problemas de optimización como el resto, sino problemas de clasificación.

Los Algoritmos Genéticos serán explicados con mayor detalle en el transcurso del proyecto.

En general, la diferencia entre unos y otros se encuentra en la representación de los individuos, los mecanismos de selección, y procreación. Entre los mecanismos de procreación, debe mencionarse dos grupos fundamentales: el de cruzamiento, que crean uno o varios individuos nuevos a partir de individuos existentes; y la mutación que operan de forma individual sobre cada individuo variando sus características en mayor o menor grado.

2.3 Principales Ventajas y desventajas de la Computación Evolutiva

Las ventajas fundamentalmente son:

- Soluciona problemas con alta complejidad.
- Es posible ver la secuencia de ejecución por pasos de forma secuencial.
- Funciona con un alto grado de paralelismo. Es decir, cada individuo se evalúa, se cruza, etc. permitiendo la utilización de ordenadores en paralelo para su ejecución.
- Los algoritmos son independientes del problema, permitiendo una gran flexibilidad, si se quiere realizar una variación no se debe reprogramar todo el algoritmo.

- Para problemas bastantes complicados, sólo se requieren una función objetivo que mida el grado de adecuación de todos los elementos del espacio de soluciones del problema.

- Son algoritmos aplicables en ordenadores de capacidad media ya que no necesitan grandes cálculos.

- Los algoritmos son fáciles de comprender y de implementar.

- Son métodos de búsqueda ciega, sólo guiadas por la función de fitness. Los operadores genéticos son independientes del problema, por esos les hacen ser generales

Las principales desventajas son:

- La implementación limita a las posibles soluciones a tener un tamaño finito el rango de las poblaciones.

- Dificultad a la hora de introducir posibles restricciones del problema.

- Elegir una función correcta de fitness. Usado una función inexacta o incorrecta puede que no sea capaz de encontrar una solución correcta al problema.

- No es unos procesos que pueda encontrar siempre un óptimo global. Al poder alcanzar una solución buena aunque no sea la óptima para la mayoría de las situaciones.

- No debería usarse para problemas resolubles sencillos porque consume mucho tiempo de ejecución y potencia computacional.

Capítulo 3. Algoritmos Genéticos

3.1 Raíces de la Teoría de la Evolución

Con el fin de facilitar la comprensión sobre los algoritmos evolutivos y sus orígenes, se presenta a continuación una breve reseña histórica sobre las raíces de la teoría de la evolución.

La teoría de la evolución a través de la selección natural fue publicada simultanéame por Charles Darwin (1809-1882) y Alfred Rusel Wallace (1823-1912) Ambos son naturalistas que invirtieron varios años coleccionando y clasificando especies en expediciones alrededor del mundo. Su experiencia los convenció que tanto plantas como animales sufren cambios a través del tiempo.

Los mecanismos de herencia fueron pobremente entendidos cuando Darwin y Wallace propusieron su teoría. Era sabido que muchos rasgos o características de los seres vivos son heredados. El austriaco Monk y Johann Mendel (1822-1884) publicaron su trabajo sobre genética en 1886 donde se demostraba que los rasgos heredados eran discretos. En 1909 Wilhelm Johanssen acuñó el término gen para referirse a una unidad discreta de herencia. El contenido completo de material genético almacenado en una célula es conocido como genoma.

3.2 Antecedentes históricos

La primera mención del término Algoritmo Genéticos (AG) se debe a Bagley (1967), que los diseñó para buscar conjuntos de parámetros en funciones de evaluación de juegos, y los comparó con algoritmos de correlación.

Pero el creador de los AG fue John Holland, que los desarrolló durante las décadas de 1960 y 1970. El propósito original no era diseñar algoritmos para resolver problemas concretos, sino estudiar, de un modo formal, el fenómeno de la adaptación tal y como ocurre en la naturaleza y extrapolar esos mecanismos de adaptación natural a los sistemas computacionales. El libro que Holland escribió en 1975, *Adaptación en Sistemas Naturales y Artificiales*, presentaba los algoritmos genéticos como una abstracción de la evolución biológica, y proporcionaba el entramado teórico para la adaptación bajo el Algoritmo Genético.

El AG de Holland era un método para transformar una población de cromosomas a una nueva población, utilizando un sistema similar a la "selección natural" junto con los operadores de cruce, mutación e inversión. La mayor innovación de Holland fue la de introducir un algoritmo basado en poblaciones con cruces, mutaciones e inversiones. Holland fue el primero en colocar la computación evolutiva sobre una base teórica firme.

Los algoritmos desarrollados por Holland al inicio eran sencillos pero dieron buenos resultados en problemas considerados difíciles. Los algoritmos genéticos establecen una analogía entre el conjunto de soluciones de un problema y el conjunto de individuos de una población natural, codificando la información de cada solución en

una cadena a modo de cromosoma, y realizando operaciones sobre esta cadena. Otra forma es operando directamente sobre los valores de las variables, sin codificar. Esto dificulta el entendimiento de estas operaciones como genéticas, y las hace más complicadas.

Las soluciones se representaban por codificaciones binarias, listas de unos y ceros. Este tipo de representaciones se utiliza incluso en problemas en donde no es muy natural. En 1985, Jong introduce las cuestiones para representar estructuras más complejas, como vectores, árboles o grafos. En la actualidad existen dos ramas del estudio.

- Utilización de strings binarios.
- Usar otro tipo de configuraciones.

Las ventajas de las primeras son que permiten definir fácilmente operaciones de recombinación, además, los resultados sobre convergencia están probados para el caso de strings binarios. La segunda puede ser más eficiente, –pero es más difícil de asemejarla a las estructuras genéticas.

Se introduce una función de evaluación de los cromosomas que se llama fitness y que está basada en la función objetivo del problema. También normalmente la población inicial se genera de forma aleatoria, aunque también se pueden utilizar métodos heurísticos para generar soluciones iniciales de buena calidad. En este caso, se debe garantizar la diversidad de la población para tener una alta representación o evitar al menos que exista la convergencia prematura.

En los últimos años, se ha desarrollado otros métodos de computación evolutiva, rompiéndose las fronteras entre algoritmos genéticos, estrategias evolutivas y programación evolutiva. Como consecuencia, en la actualidad, el término "algoritmo genético" se utiliza para designar concepto de Holland.

3.3 Características generales de los Algoritmos Genéticos

Como se ha comentado, los algoritmos genéticos son algoritmos basados en mecanismo de evolución genéticos natural, en los que se introducen los conceptos de supervivencia, y selección. Los AG son superiores a una búsqueda aleatoria ya que explotan la información histórica para mejorar el rendimiento de la búsqueda. Normalmente son utilizados para optimizar funciones y pueden aplicarse a muy diversos problemas.

Las características de los AG:

- Los AG buscan en una población de soluciones.
- Los AG sólo utilizan la información de una función objetivo o función fitness y no incorpora ningún otro conocimiento específico.
- Los AG usan reglas de transición probabilísticas, no reglas determinanticas.
- Los AG sólo puede trabajar con poblaciones finitas.

Los AG necesitan una población con un número determinado de cadenas de longitud finita sobre un alfabeto finito. A estas cadenas se lo denomina cromosomas. Asociado a cada posición de la cadena hay un gen. Esto es un conjunto que contiene los símbolos que pueden aparecer en cada posición de la cadena. Cada uno de los elementos de la cadena se llama alelos (Michalewicz 1996).

Los AG ocupan un lugar central dentro de la Computación Evolutiva, debido a las siguientes razones:

- En los AG se reúnen de un modo natural todas las ideas en que se basa la Computación Evolutiva.
- Son muy flexibles, porque pueden adoptar con facilidad nuevas ideas generales o específicas en la Computación Evolutiva.
- En los AG se reúne la mayor base teórica de entre las técnicas de la Computación Evolutiva.
- Son los algoritmos que necesitan menos conocimiento para su funcionamiento. Lo que no impide que puedan ser utilizados con conocimiento específico, teniendo que realizar para ello un mínimo esfuerzo adicional en la modificación de algunos aspectos del algoritmo, como en los operadores de cruce y mutación.
- Pueden ser utilizados en computadores de capacidades medias dando buenos resultados, para problemas de una alta complejidad.

Las características de la búsqueda de los Algoritmos Genéticos son:

- Ciega porque no disponen de ningún conocimiento específico del problema, salvo valores de la función objetivo.
- Codificada puesto que no trabaja con el dominio del problema, sino sobre su representación de sus elementos.
- Múltiple porque busca simultáneamente entre un conjunto de candidatos
- Estocástica porque intervienen componentes aleatorios.

Los AG son técnicas de búsqueda global, con una capacidad para explorar y explotar un espacio de soluciones dado, utilizando las medidas de rendimiento disponibles. Además, los operadores genéticos tales como el cruce, mutación y reproducción permiten simulaciones relativamente rápidas de un proceso amplio de aprendizaje de la naturaleza. Los AG trabajan sobre una cadena codificada y no directamente en los parámetros.

En muchos problemas, no hay garantía de suavidad y unimodalidad en el espacio de soluciones. Las técnicas de búsqueda tradicionales a menudo fallan en tales espacios de búsqueda. Los AG son capaces de encontrar soluciones próximas al óptimo en espacios de búsqueda complejos.

Para aplicar los algoritmos genéticos en la resolución de un problema hay que determinar un código genético para el problema y una función de fitness que asigna un valor a cada cromosoma. Los AG emplean típicamente tres operadores: selección, cruce

y mutación. Estos operadores se aplicarán de generación en generación y hasta que no cumpla unas condiciones de parada, seguirán aplicándose dichos operadores sobre la población de individuos. Habrá individuos que se ajustan más que otros a la solución óptima. Los individuos que se ajustan mejor a la solución poseen mayor posibilidad de tener descendencia. Los hijos contendrán información genética de los padres e información aleatoria, gracias a los operadores de mutación.

3.4 Tipos de codificación

Un individuo representa una solución al problema y vendrá definido por las variables del problema. Debe elegirse la codificación adecuada para el problema a resolver. Cada individuo está formado por una secuencia finita de genes. Para la representación de los alelos se suelen utilizar números naturales. Se debe fijar los símbolos a utilizar, de modo que cada gen puede especificarse por un único número, conocido como cardinalidad del gen.

La codificación elegida para representar las soluciones las siguientes propiedades:

- Completitud: Todas las posibles soluciones deben poder ser representados.
- Coherencia: Únicamente se deben representar soluciones factibles del problema
- Uniformidad: Todos los objetos deben estar representados por la misma cantidad de codificaciones, no deben existir objetos sobre representados ni sub-representados.
- Localidad: Pequeños cambios en los individuos se han de corresponder con pequeños cambios en las soluciones.

En el contexto de AG, existen dos tipos fundamentales de codificaciones: codificación binaria y codificación real o entera, las cuales se describen a continuación.

Codificación Binaria

En este caso lo que se hace es codificar los valores de las variables a código binario y es la representación más habitual que se da en las técnicas evolutivas. Una vez realizado esto, es más sencillo su tratamiento ya que es más intuitivo el diseño y funcionamiento de los operadores genéticos, al ser el resultado muy similar a lo que podría ser un cromosoma real.

De esta manera, la generación de una población usando codificación binaria seguiría el siguiente proceso:

- Seleccionar las variables que describen la solución. Cada una de estas variables se va a codificar en un cromosoma.

- Elegir la forma de codificar en términos binarios cada variable
- El conjunto de variables codificadas forma el genoma del individuo o cromosoma.

Este mismo genoma, una vez decodificado a los valores originales de las variables representa una posible solución. El genoma descodificado se denomina fenotipo

En el algoritmo clásico, el valor de cada variable se codifica de decimal a binario

Codificación Real o Entera.

Existen algunas ventajas prácticas en la utilización de valores reales o enteros para codificar los individuos de un AG. Mejora la eficiencia ya que no es necesario descodificar cada parámetro para evaluar la función objetivo, de esta manera se puede ahorrar tiempo. Se utiliza menos memoria, ya que pueden utilizarse arreglos de variables de punto flotante para representar un cromosoma en el caso codificación real. Y en el caso de codificación entera usan variables enteras.

Para este tipo de representaciones, los operadores genéticos deben ser de tal manera, que operan sobre los valores reales o enteros. Los operadores de cruce en estas codificaciones son muy similares al de AG con codificación binario excepto que sus puntos de cruce deben estar entre los genes. En cambio el de mutación es un método aleatorio, cambiando un gen por un número aleatorio que esté dentro de un intervalo seleccionado.

3.5 Función de fitness o de evaluación

La función de fitness o función de adecuación se debe elegir cuidadosamente y viene dada por el problema a resolver. Dado un cromosoma particular, la función de fitness devuelve un valor numérico, proporcional a la utilidad o capacidad del individuo en cuanto a resolver el problema planteado.

En algunos casos, como cuando se pretende maximizar una función, es obvio que la función fitness debe devolver el valor de la función a optimizar, ya que los AG estándar están diseñados para resolver problemas de maximización. Pero no es siempre este el caso, ya que pueden existir problemas en los que se quieren optimizar diversas características, maximizando unas y quizás minimizando otras. Un ejemplo sería el diseño de una construcción, en el que se pretende optimizar variables de resistencia/peso, coste, tiempo construcción, número de trabajadores, etc., siendo diferente el criterio de optimización de cada una. En estos casos es necesario definir una única función que aglutine a todas.

Si se trata de minimizar una función, se debe realizar los siguientes pasos: Elegir un valor A que sea superior a los posibles valores de la función. Entonces si se quiere obtener los valores mínimos, se maximizará la siguiente función:

$$F(x) = A - f(x) \qquad \text{con } f(x)<A$$

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

Una de la maneras de controlar la diversidad de las aptitudes es definiendo una función de aptitud neta, distinta de la función de evaluación. Esto es lo que se conoce como escalado de la función de evaluación que trata de evitar un fenómeno con causas opuestas al anterior pero de similares efectos: es el fenómeno de la convergencia prematura, que hace que se obtengan superindividuos. Desde el estudio pionero de De Jong, el escalado de los valores de la función objetivo se ha convertido en una práctica ampliamente aceptada. Se hace esto para mantener los niveles apropiados de competición a lo largo de una simulación. Sin escalado, en las fases tempranas hay una tendencia a que unos pocos individuos dominen el proceso de selección (se llamen superindividuos). En este caso, los valores de la función objetivo deben ser subescalados para evitar que tales superindividuos ocupen la población. Más tarde, cuando la población ha convergido casi en su totalidad, la competición entre los miembros de la población es menos intensa y la simulación tiende a evolucionar erráticamente. En este caso los valores de la función objetivo deben ser sobreescalados para acentuar la diferencia entre los miembros de la población con el fin de continuar favoreciendo a los mejores miembros.

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

3.6 Componentes de los Algoritmos Genéticos

Para la utilización de los AG se utilizará la estructura cuyo nombre es cromosoma. Los cromosomas constituyen una concatenación de un número de subcomponentes denomina genes. Los genes se encuentran en una posición del cromosoma denominadas loci y toman un conjunto de valores conocidos como alelos. En una representación binaria un gen es un bit. El término biológico genotipo se refiere a la composición genética de un individuo y corresponde a la estructura descodificada en AG. La función de fitness asignará a cada individuo un valor de adecuación del individuo al problema a resolver

Un Algoritmo Genético simple genera de forma aleatoria una población inicial de n individuos y realiza un número fijo de generaciones (o hasta que algún criterio de parada) para ir modificando la población de individuos. Durante cada generación, el AG lleva acaba una selección proporcional al fitness, seguida de un cruzamiento para finalizar con el proceso de mutación.

La población

La población está compuesta por un número determinado de individuos, cada uno de los cuales representa una posible solución del problema que se está tratando. La longitud de la cadena de los individuos depende del esquema de codificación, del número y tipos de parámetro que el Algoritmo Genético tiene que optimizar.

Debido a que las soluciones proporcionadas por un AG depende de la información contenida en la población, hay dos factores importantes a los que se debe prestar una especial atención. Son el tamaño de la población y la aleatoriedad de la población inicial. La diversidad de soluciones aumenta con la diversidad de la población, mientras se reduce la convergencia a un óptimo local. En cambio una población formada por muchos individuos incrementa el coste computacional. La población inicial debe ser aleatoria y distribuida por todo el espacio de la búsqueda.

Para poder desarrollar el proceso genético correctamente los individuos deben pertenecer a un conjunto cerrado denominado espacio de búsqueda. Una pequeña dificultad para la resolución del problema sería la de acotar el espacio en el que encontrar la solución del problema. Aunque no se conozca con exactitud la solución, siempre es muy útil acotar el espacio de las posibles soluciones.

Otro punto que influye en la resolución de los algoritmos genéticos es el tamaño de los individuos. Suele venir dado como una consecuencia de los criterios de representación y codificación del problema. No se debe alagar innecesariamente la longitud del individuo, evitando todas las posibles redundancias.

Además, cada individuo tiene asociado un valor de fitness que indica lo bueno o lo malo que es el individuo respecto a la solución real del problema. Siendo el valor del fitness mayor más se asemejará a la solución deseada.

Procedimiento básico

El proceso que realiza un algoritmo genético es el siguiente:

1. El proceso comienza con una población que consta de número de individuos, que podrán ser elegido a través de un fichero o de forma aleatoria

2. Se evalúa cada uno de los individuos utilizando la función de fitness

3. Si se ha obtenido el óptimo con esa población se termina la ejecución, sino se continua la ejecución

4. Dicha población será sometida a un proceso de selección para constituir una nueva población intermedia de individuos. De dicha población se extrae un grupo de individuos llamados progenitores, que se van a reproducir.

5. Los progenitores de alteración y recombinación en la fase de reproducción en virtud de las cuales se generan nuevos individuos que constituyen la descendencia. A través de métodos de Cruce y selección

6. Se forma una nueva población, seleccionado los supervivientes y las descendencias.

7. Se vuelve ejecutar el algoritmo desde el punto 2.

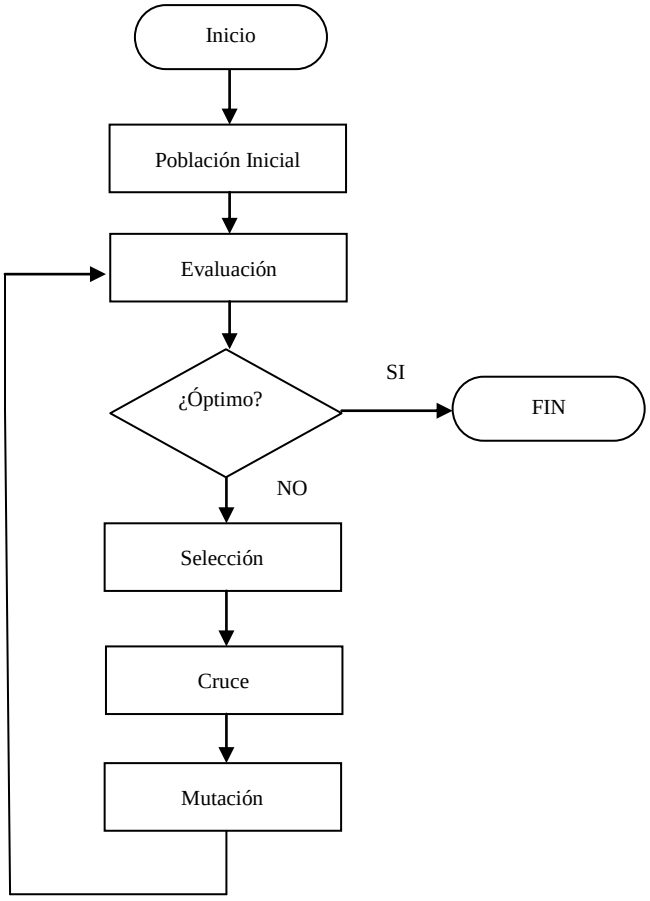


Figura 3.1. Procedimiento básico de un Algoritmo Genético

Inicialización

La inicialización son todas las operaciones previas que se deben realizar para poder continuar con el resto de acciones. Estas operaciones son, entre otras, la reserva de memoria para la población y la inicialización de sus individuos e inicialización de otros parámetros y variables auxiliares (como la identificación de los métodos genéticos que van a ser usados). La población puede inicializarse de forma aleatoria o a través de un fichero. Conviene que la población contenga la mayor cantidad posible de puntos factibles. En ocasiones es difícil obtener puntos factibles y se deberá iterar con una

cantidad muy reducida de ellos. Si esto ocurre, no debería iniciarse con una población pequeña, perjudicando el requisito de diversidad, ni tampoco se debe introducir una gran cantidad de puntos, ya que se reduce la eficiencia del AG. La población inicial es uno de los elementos más importantes que harán que el método converja más o menos rápido a la solución deseada.

Evaluación

El siguiente paso es evaluar utilizando la función de fitness todos los individuos de la población, de modo que cada individuo tenga un valor de adecuación al problema.

Selección

La selección se lleva a cabo por la medida de adecuación de los individuos al entorno, Back 1994.Su finalidad es un proceso en el cual se escogen los individuos que se van a tener en cuenta para la siguiente acción (cruce) y que por la tanto van a dar lugar a la descendencia, que conformara la siguiente generación. Hay varios métodos para realizar esta selección. Normalmente aquellos individuos con mayor fitness, tienen mayor probabilidad de ser escogidos durante el proceso de selección, de esta manera se favorece que prevalezcan las características de los individuos con mayor fitness.

El método más utilizado es el directamente proporcional a la medida de adecuación, conocida como método de la ruleta o proporcional a la adecuación, que se explicará a continuación. Otros diseños emplean modelos en los que tiene lugar un cierto grado de selección al azar de individuos que forma un subgrupo de selección. Cada método de selección asocia una distribución de probabilidad de selección a cada individuo de la población, por tanto, cada método introduce un sesgo de búsqueda. Se establece una clasificación dependiendo de cómo se aplica el operador de selección, en función de la probabilidad de supervivencia del conjunto n mejores individuos. Si la probabilidad es igual a 1 entonces a la población se denomina "solapada". Aquellos individuos seleccionados como "solapados" no sufren ninguna alteración. En las poblaciones "no solapadas" existe el problema que los individuos mejores pueden que no aparezca en las siguientes generaciones. Los mecanismos de muestreo son muy variados, distinguiendo tres grupos fundamentales según el grado de intervención del azar en el proceso:

- Muestreo directo: Se toma un subconjunto de individuos de la población siguiendo un criterio fijo, del estilo de "los k mejores", "los k peores", "elección a dedo",

- Muestreo aleatorio simple: Se asignan a todos los elementos de la población base las mismas probabilidades de formar parte de la muestra.

- Muestreo estocástico: Es el método de selección más usado en los algoritmos genéticos. Se asignan probabilidades de selección o puntaciones a los elementos de la población base en función (directa o indirecta) de su aptitud.

Selección por Ruleta

Este método asigna descendencia y se basa en la relación del fitness de una cadena, respecto al valor del fitness medio de la población. Primero el sistema asigna a cada individuo i de la población una probabilidad de selección *p(i)*, de acuerdo a la razón entre el i-ésimo fitness y el fitness total acumulado de la población:

$$P(i)= fitness(i)/finess\_total$$

El siguiente paso es calcular la probabilidad acumulativa, *Q(i)*, realizando la suma de cada una de las *P(i)* calculadas anteriormente:

$$Q(i) = \sum P(j)$$

De esta forma el individuo que posea un fitness alto, tendrá una porción mayor en la ruleta, en cambio los individuos con fitness más bajo, poseerán una porción menor. Para seleccionar un individuo se genera un número aleatorio, de manera que se elige el elemento i-ésimo de la población.

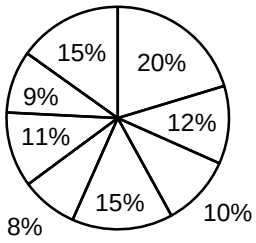
Algoritmo de la ruleta:

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

La selección a través del método de la ruleta es intuitiva y fácil de implementar. Sin embargo, existen problemas de escalado, pueden manejar únicamente problemas de maximización, causan grandes errores de muestreo, y requieren un tamaño de población grande.

Un ejemplo sería el siguiente:

Números	Variable	Fitness	Porcentaje
1	11110000	30.5	20.33 %
2	11001100	17	11.33 %
3	11110011	15.5	10.33 %
4	10010110	22	14.66 %
5	10101010	12.3	8.20 %
6	01010101	16.4	10.93 %
7	10011010	13.6	9.06 %
8	11100110	22.7	15.13 %
		150	



Ahora deberá girar la ruleta tantas veces que sea necesario. Si la población es no solapada debe elegirse tantos individuos como la forman la población.

Selección por Batallas o Torneo

El método de selección por torneo (Brindle,1981; Gooldberg & Deb, 1991) realiza un número n de torneos y selecciona un número n individuos. En cada torneo compiten *q* elementos de la población y es elegido el más adaptado, o el que presente mayor fitness.

Este método de selección funciona correctamente para el manejo de problemas de maximización y minimización y puede ser fácilmente extendido a problemas con múltiples criterios. Existen varias opciones en función del número de individuos que formen parte del Torneo. En binario, los pares de individuos se escogen al azar de la población. El que tenga mayor fitness se marcará como seleccionado.

Una desventaja del método consiste en que el individuo con menor fitness nunca será elegido porque siempre perderá el torneo. En cambio el individuo con mayor fitness siempre ganará los torneos. Con este método nunca se reproducirá los individuos de menor fitness.

Este método conviene utilizarlo cuando el tamaño de la población sea muy alto. Para el método de la ruleta, si la población es muy grande, el método resulta

computacionalmente muy costoso, mientras que en la batalla no importa lo grande que sea la población, puesto que los individuos son elegidos con aleatoriedad.

Algoritmo de la Batalla o Torneo

1. Escoger tamaño de torneo q, normalmente se suele elegir q = 2.
2. Crear una permutación aleatoria de M enteros.
3. Comparar la adaptación de los próximos q-miembros de la población y seleccionar el mejor.
4. Si se finalizó la permutación, debe generarse una nueva permutación.
5. Repetir el proceso hasta llenar la población.

Un ejemplo sencillo será siguiente:

Números	Variables	Fitness
1	11110000	30.5
2	11001100	17
3	11110011	15.5
4	10010110	22
5	10101010	12.3
6	01010101	16.4
7	10011010	13.6
8	11100110	22.7

Números	Variables	Fitness
1	11110000	30.5
4	10010110	22
6	01010101	16.4
8	11100110	22.7

Números	Variables	Fitness
1	11110000	30.5
2	11001100	17
3	11110011	15.5
4	10010110	22
5	10101010	12.3
6	01010101	16.4
7	10011010	13.6
8	11100110	22.7

Números	Variables	Fitness
1	11110000	30.5
1	11110000	30.5
4	10010110	22
4	10010110	22
7	10011010	13.6
6	01010101	16.4
8	11100110	22.7
8	11100110	22.7

Selección por Rango

Cada individuo es seleccionado proporcionalmente al rango de su función objetivo y no a la magnitud que toma ésta. El algoritmo de selección por rango es como sigue:

1. Ordena la población del mejor individuo x = 1 al peor x=M.
2. Se transforma linealmente este ordenamiento:

   $$G(x)= A\ r(x) + B$$

   Donde A y B son constantes positivas a determinar y r(x) es el rango (orden) obtenido en punto 1.
3. Se debe completar la población de forma aleatoria según resultado del 2.

Este método impide que en las primeras generaciones se creen solamente "superindividuos" inundando la población. Por otro lado, en las últimas etapas del algoritmo, por la difusión en la población, los individuos más adaptados tienen prestaciones parecidas. Al atender al rango y no a la función objetivo se favorece de forma sostenida a los mejores individuos, aun cuando la diferencia con el resto no es muy notable.

Es más complejo de implementar que el método de la ruleta.

Números	Variables	Fitness
1	11110000	30.5
2	11001100	17
3	11110011	15.5
4	10010110	22
5	10101010	12.3
6	01010101	16.4
7	10011010	13.6
8	11100110	22.7

Ordenar →

Números	Variables	Fitness	Rango
1	11110000	30.5	1
8	11100110	22.7	2
4	10010110	22	3
2	11001100	17	4
6	01010101	16.4	5
3	11110011	15.5	6
7	10011010	13.6	7
5	10101010	12.3	8

$$G(x)= 2x-4;$$

		Números	Variables	Fitness	Rango
1	-2	No se rellena ahora			1
2	0	No se rellena ahora			2
3	2	4	10010110	22	3
4	4	2	11001100	17	4
5	6	6	01010101	16.4	5
6	8	3	11110011	15.5	6
7	10	No se rellena ahora			7
8	12	No se rellena ahora			8

Y queda el tercer paso que es rellenar la población de forma aleatoria hasta completar la población

Números	Variables	Fitness
4	10010110	22
2	11001100	17
6	01010101	16.4
3	11110011	15.5
7	10011010	13.6
2	11001100	17
1	11110000	30.5
8	11100110	22.7

Selección Elitista

El método de selección elitista (1975, De Jong) asegura que los mejores elementos de una población sobrevivirán de generación en generación. La estrategia más básica de elitismo consiste en copiar el mejor elemento de la población actual directamente en la siguiente población, si este elemento no ha sido trasferido a través del proceso normal de selección, cruce y mutación. En general cualquier método estándar de selección puede convertirse en uno elitista.

De esta forma se garantiza que los mejores individuos no se van a perder producto de la manipulación genética de los otros operadores. Este operador ha sido incluido en el algoritmo implementado donde el número de individuos seleccionados en forma directa está determinado por un porcentaje ingresado por el usuario. La utilización de porcentajes elevados puede generar una convergencia prematura del algoritmo.

Por otro lado, sin ninguna medida adicional, este operador puede conducir a la aparición de clones. A pesar de estas circunstancias, el elitismo presenta interesantes ventajas, siendo utilizados con frecuencia en aplicaciones reales.

Algoritmo Elitista

1. Evaluar los individuos de la población

2. Escoger el número que sobrevivirá N.

3. Introducir los N mejores individuos para las siguientes poblaciones

Números	Variables	Fitness
1	11110000	30.5
2	11001100	17
3	11110011	15.5
4	10010110	22
5	10101010	12.3
6	01010101	16.4
7	10011010	13.6
8	11100110	22.7

Elegir los 4 mejores →

Números	Variables	Fitness
1	11110000	30.5
8	11100110	22.7
4	10010110	22
2	11001100	17

Epidemia

Este operador elimina todas las soluciones excepto las que poseen mejor desempeño o valor de fitness. En el algoritmo el número de soluciones que debe ser salvado puede ser especificado, así como también la ocasión entre generaciones en que este operado se va a aplicar. Las soluciones eliminadas son entonces reemplazadas por nuevas generadas con el fin de refrescar la información procesada por el algoritmo.

Cruzamiento

El cruce es un operador de reproducción que proporciona intercambio de información aleatoria. Consiste en generar dos nuevos individuos (hijos) a partir de otros dos individuos (padres) seleccionados. Los cromosomas de los padres deben de ser de la misma longitud. Este es el mecanismo básico para crear la nueva población, la siguiente generación, a partir de la antigua.

El cruce debe realizarse de forma que los dos individuos seleccionados para cruzarse intercambien parte de su contenido, dando así como resultado dos nuevos individuos formados por parte de cada uno de los padres. Los puntos de cruce se eligen de forma aleatoria y puede ser que sólo sea un punto, dos o n-puntos.

Los nuevos individuos creados pasarán a formar parte de la población, mientras que los padres se eliminarán y se pasarán a coger dos nuevos padres. El proceso se hace hasta que la lista de padres quede vacía. Si el número de padres es impar, el padre que quede si pareja pasará directamente a la siguiente generación.

El valor de la tasa de cruce se deberá especificar según el tipo del problema y tendrá que determinarse empíricamente. El método de cruce realiza una búsqueda más rápida que la mutación para funciones que implican interacciones no lineales entre los bits de la cadena. Si la tasa de cruce de un gen es dependiente de su posición en la cadena, se dice que le operador de cruce tiene sesgo posicional. En cambio si la distribución del número de genes intercambiados no es uniforme, el operador tiene un sesgo distribucional.

Tanto la selección como el cruce de los individuos seleccionados se repiten iterativamente hasta generar una nueva población del tamaño especificado (el mismo que la población inicial)

Hay dos formas de aplicar el algoritmo cruzamiento:

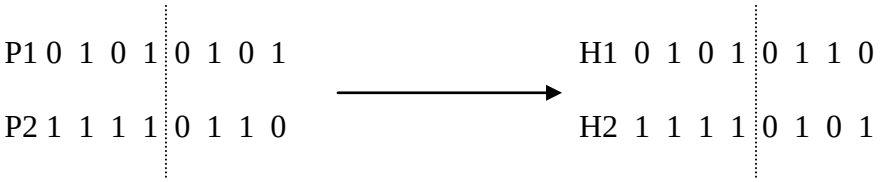
- Se quieren reproducir a los mejores individuos, sin importar los de bajo fitness, , como consecuencia se crea súper-individuos, reduciéndose la posibilidad de encontrar nuevos individuos al viciarse la población, ya que los individuos se igualan genéticamente.

- La pareja se elige entre toda la población de forma aleatoria. De esta forma los individuos tienen posibilidades de sobrevivir y reproducirse aunque no posean un alto fitness. Se evita así la degeneración de la población.

Este operador realiza una búsqueda con profundidad en el espacio soluciones. La idea es que la solución no se alcanza como un todo, si no por la recombinación, es la teoría de los llamados "bloques constructivos" Goldberg 1989, Stephens 1999.

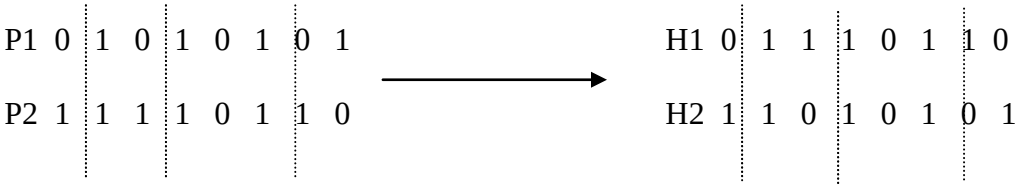
Cruce en codificaciones de binarios

La forma más sencilla de cruce para codificaciones binarias es el siguiente: se elige un punto aleatorio, por ejemplo situado entre el tercer y cuarto bit, y se generan hijos uniendo las diferentes partes, como se muestra en el ejemplo.



Cruce multipunto:

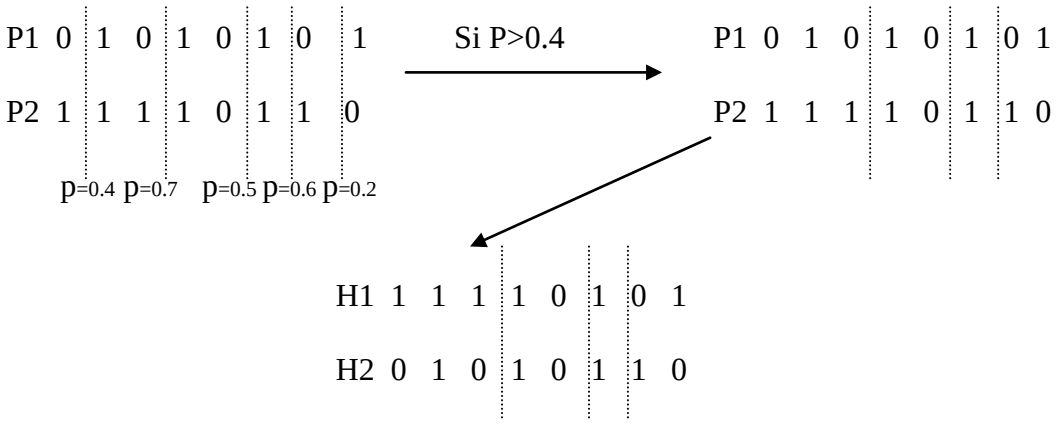
Consiste en cruzar los individuos a dos o más puntos. Los puntos elegidos al azar se encuentran entre el primer y segundo, el tercer y cuarto y el sexto y séptimo.



Cruce segmentado:

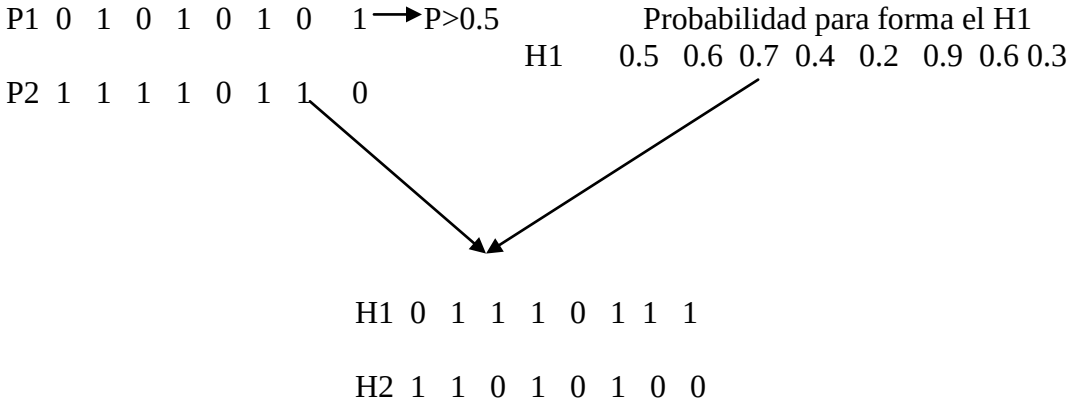
Es una versión del cruce multipunto que permite la introducción de variabilidad en el número de puntos de cruce. Consiste en reemplazar el número fijo de puntos de cruce por una probabilidad de segmentación que da cuenta de la posibilidad de que se produzca un cruce al llegar a cierto punto de la cadena.

En este ejemplo tenemos unos puntos aleatorios asociados a una probabilidad y también poseemos una probabilidad segmentada. Se eliminan aquellos puntos de corte que no sea superior a la probabilidad.



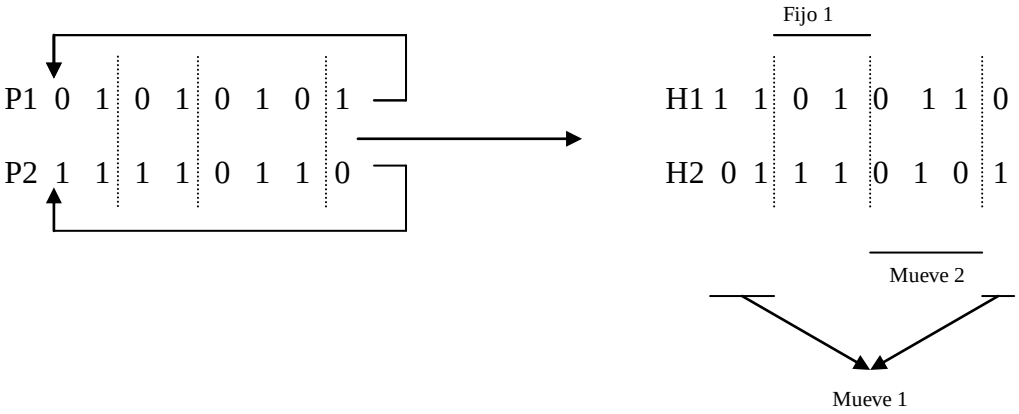
Cruce uniforme:

Para cada bit del primer descendiente se decide con cierta probabilidad qué progenitor heredará su valor en esa posición. Y el segundo descendiente recibe el correspondiente gen de otro progenitor. El cruce uniforme cambia genes en vez de bloques.



Cruce anulares:

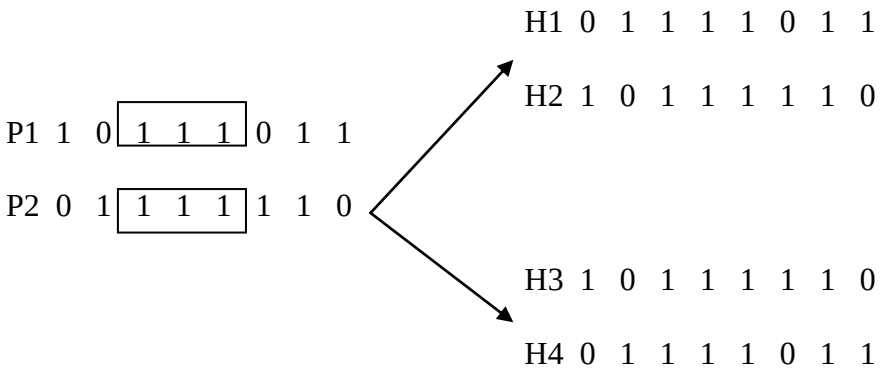
Se considera que el primer y el último bit de la cadena son adyacentes, de ahí la forma de anillo.



Si fuera un método normal los cromosomas estarían formados en este caso por cuatro regiones que lo formarían el 1,2 gen la primera zona; el 3 y 4 la segunda zona; 5,6 y 7 la tercera zona y la última zona sería el 8. Pero si se usa el método de cruce anulares sólo existen tres regiones que las forman, el 1,2, 8 genes la primera zona; el 3 y 4 la segunda zona y la última zona la forma 5,6 y 7.

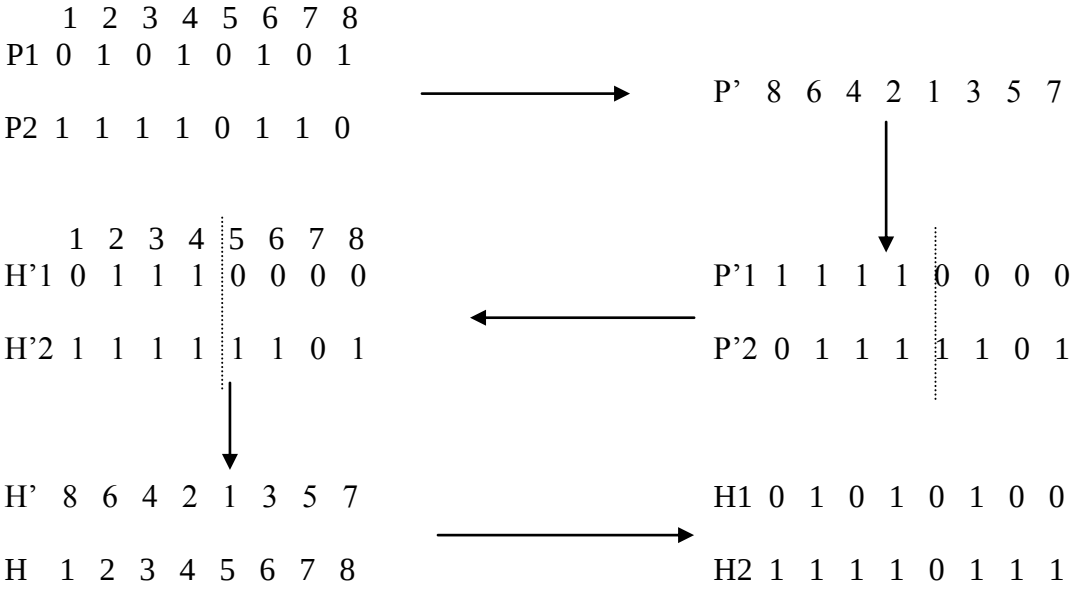
Cruces externos:

Independiente de la posición y del número de bits que ocupe cada gen en la estructura de los individuos, el cruce de los algoritmos genéticos se realiza en el ámbito de bits. Sin embargo, en algunos problemas es más natural obligar a que los cruces se produzcan exclusivamente en los puntos de separación entre genes.



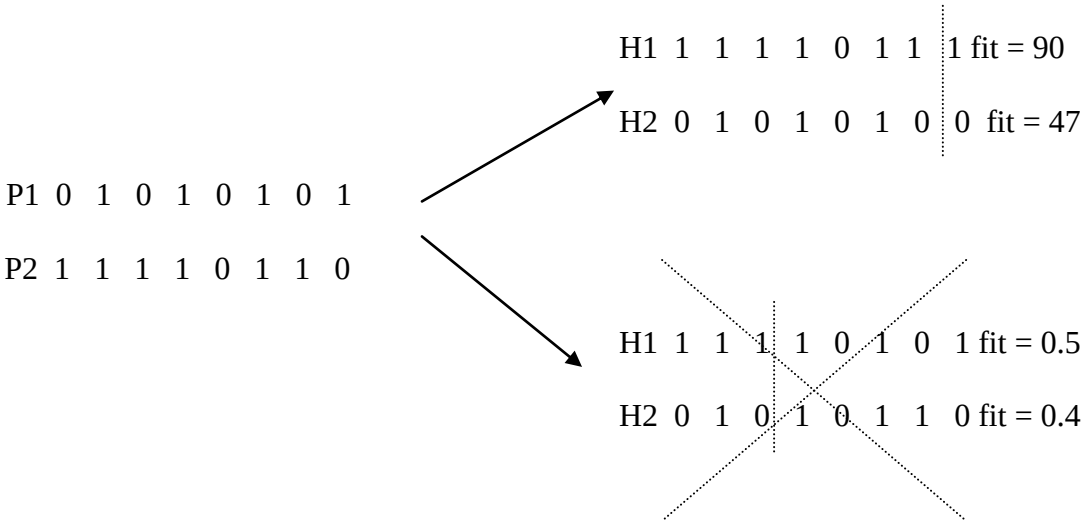
Cruces con barajado:

Antes de realizar el cruce se desordenan las posiciones del cromosoma de forma aleatoria de ambos progenitores a la par. Después se realiza el cruce y finalmente se deshace el barajado, colocándose los genes en su posición original.



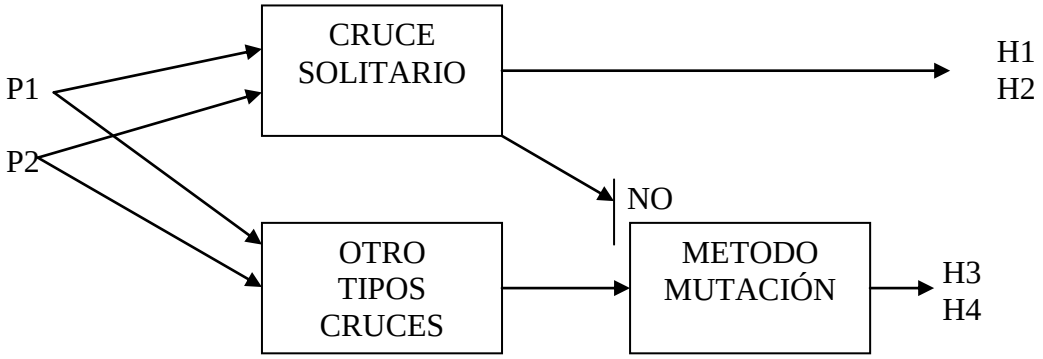
Cruces con adaptación:

Se sustituyen las estrategias de asignación aleatoria de los puntos de cruce por estrategias de adaptación para la asignación de dichos puntos, en base a la función de fitness. Se hace esto con la esperanza de que los puntos de cruce experimenten un proceso de evolución, es decir, si cierto punto ocasiona descendencia de baja calidad, se extingue, y en caso contrario prospera.



Cruces solitarios:

Trata de prohibir la aplicación de mutaciones a individuos procedentes de cruces. Esto introduce orden en el proceso de búsqueda y elimina la asimetría de aplicación entre ambos operadores.



Cruce en codificaciones de reales

Cruce por un Punto:

Opera de formar idéntica al caso de codificación binaria, pero ahora lo que se permuta son componentes del vector en lugar de bits. Con este ejemplo se explica el procedimiento.

P1: (0.9	0.7	0.5	0.8	0.4)		H1: (0.9	0.7	0.6	1.0	0.2)
P2 :(1.2	1.3	0.9	1.0	0.2)		H2 :(1.2	1.3	0.5	0.8	0.4)

Cruce Aritmético:

Los hijos serán combinaciones lineales de los dos vectores padres, pudiendo introducir constantes. También es frecuente introducir aleatoriedad en los operadores. Un ejemplo muy sencillo seria

P1: (0.9 0.7 0.5 0.8 0.4)

P2 :(1.2 1.3 0.9 1.0 0.2)

H1: P1 - P2 + 1

H2: P2 + P1 – 1.5

H1: (0.7 0.4 0.6 0.8 1.2)

H2: (0.6 0.5 -0.1 0.3 -0.9)

Cruce en codificación de enteros

Cruce Parcialmente Mapeado:

El hijo se construye eligiendo un segmento de la lista de unos de los padres, preservando el orden y la posición del mayor número posible de elementos del otro padre.

Con el siguiente ejemplo se explica el procedimiento:

P1: (9	7	5	3	1	2	4	6	8)	
P2 : (2	4	6	8	1	3	5	7	9)	

Puntos de cortes

Se elige los puntos cortes de forma aleatoria. Producirán dos hijos, incompletos intercambiando sus segmentos centrales:

H1: (X X | 6 8 1 3 5 | X X)

H2 :(X X | 5 3 1 2 4 | X X)

Deben rellenarse los hijos con los valores de los padres, siempre que no exista conflicto con los valores introducidos ya en la lista:

H1: (9 7 | 6 8 1 3 5 | X X) 6,8 no se puede poner

H2 :(X X | 5 3 1 2 4 | 7 9) 2,4 no se puede poner

Para finalizar se completan con los valores restantes atendiendo a la asociación de los padres

H1: (9 7 | 6 8 1 3 5 | 2 4)

H2 :(6 8 | 5 3 1 2 4 | 7 9)

Cruce Cíclico:

Los hijos se construyen teniendo en cuenta que cada posición proceda de uno de los padres. Se muestra un ejemplo:

P1: (9 7 5 3 1 2 4 6 8)

P2 :(2 4 6 8 1 3 5 7 9)

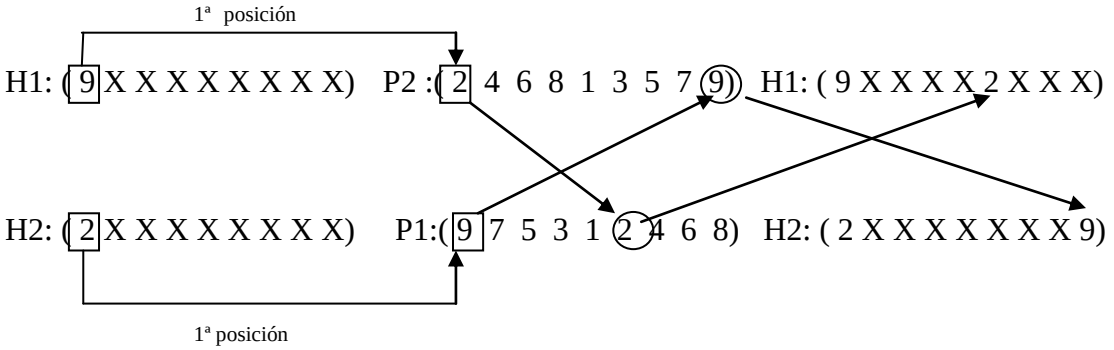
Se comienza tomado una posición al azar (por ejemplo la primera) Esta posición se mantiene en la siguiente generación.

H1: (9 X X X X X X X X)

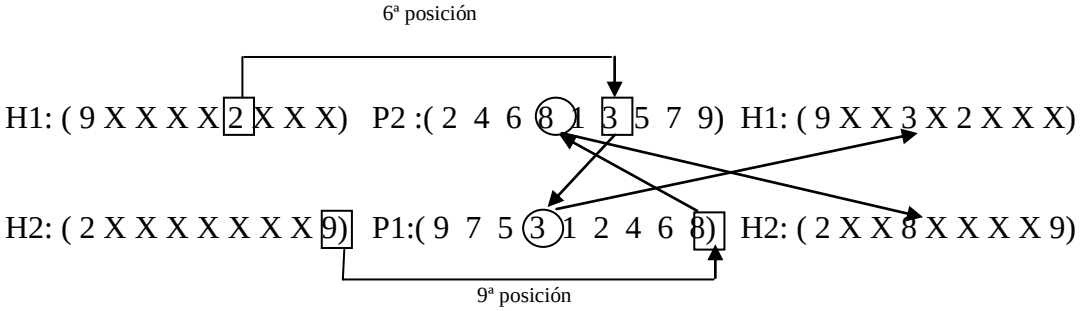
H2: (2 X X X X X X X X)

Se obtiene la posición de la última selección H1 en este caso es la 1ª posición, se busca esa posición en el padre P2 obteniendo su valor en este ejemplo sería el "2", Ese valor se mantiene en la siguiente generación en la posición donde se encuentra en P1 en este caso sería la posición 6ª.

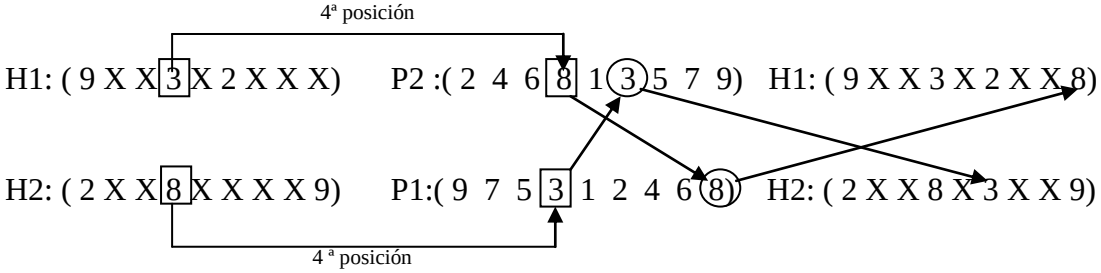
Con el H2 sería realizar los mismos pasos, se selecciona el valor de la 1ª del P1 en este caso sería el valor "9" Y se mantendría ese valor en la posición del P2 en este caso sería en la 9ª



Ahora se volverá a repetir el paso anterior. Se mira la posición de la última selección del padre opuesto obteniendo su valor. A continuación se busca dicho valor en su padre pasando ese valor en la posición que tiene el padre. En este caso el último valor seleccionado del H1 es 2 que está en la posición 6ª. En el P2 la posición 6ª el valor lo ocupa el número 3. Se busca en el H1 el valor 3 que se encuentra en la posición 4ª y se copiará ese 3 en la posición 4ª al H1. Con el H2 se realiza el mismo proceso en este caso el último valor seleccionado era el 9 que está en la 9ª posición. Se busca que valor del P1 está en la 9ª es el 8. Y se copia ese valor en la posición donde la tiene su padre



Ahora H1 el valor es 8 y la posición es la 9. El siguiente paso en el H1 sería buscar el valor 9, pero como ya está en la lista se completa el ciclo. En H2 el valor para poner en la lista es 3 y su posición y 6. Y como en el H1 también se completa el ciclo porque el siguiente valor sería el 2 que ya está en la lista.



Como se han completado los ciclos, el resto de los valores son seleccionados del otro padre. Es decir, el H1 se completa con el padre P2 y el H2 con el P1.

H1: (9 4 6 3 1 2 5 7 8)

H2: (2 7 5 8 1 3 4 6 9)

Cruce de Ordenado Uniforme:

Preserva parte de la información de uno de los padres e incorpora información del otro según el orden de los valores. Con el siguiente ejemplo se comprenderá mejor.

P1: (9 7 5 3 1 2 4 6 8)

P2 :(2 4 6 8 1 3 5 7 9)

Debe generase una máscara del mismo tamaño, que los padres

m: (0 1 0 1 0 1 0 1 0)

Se copian los valores de su padre cuando en la máscara aparece un 1. En este caso los hijos quedan de la siguiente forma:

H1: (X 7 X 3 X 2 X 6 X)

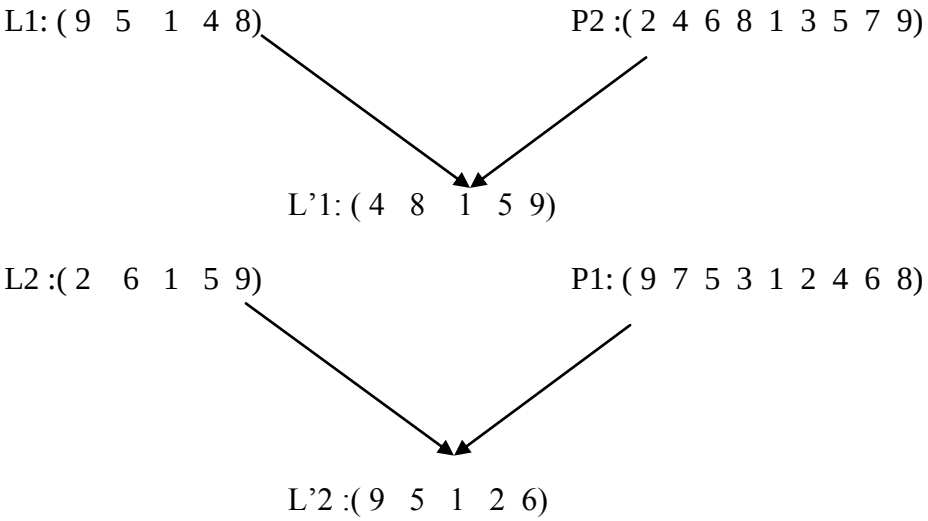
H2 :(X 4 X 8 X 3 X 7 X)

Se genera una lista con los valores que no se han introducido en los hijos.

L1: (9 5 1 4 8)

L2 :(2 6 1 5 9)

Se modifica la lista en el orden que aparece en el otro padre.



Se incluye el orden de la lista en los hijos.

L'1: (4 8 1 5 9) + H1: (X 7 X 3 X 2 X 6 X) = H1: (4 7 8 3 1 2 5 6 9)

L'2 :(9 5 1 2 6) + H2 :(X 4 X 8 X 3 X 7 X) = H2: (9 4 5 8 1 3 2 7 6)

Mutación

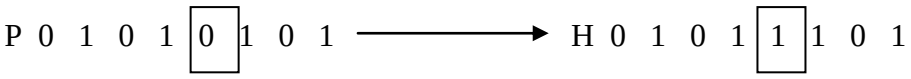
La mutación se ve como un operador reproductivo que devuelve involuntariamente valores de genes perdidos. Es un proceso totalmente aleatorio que consiste en alterar los componentes de alguno de los individuos. De esta manera se consigue expandir una pequeña parte de la población hacia zonas inexploradas del espacio de búsqueda, lo que favorece la variedad genética y permite que el algoritmo salga de óptimos locales.

Obviamente, debe existir un equilibrio entre la probabilidad de mutación y el efecto de herencia por cruce. La probabilidad de que un individuo sufra una mutación debe ser relativamente pequeña de tal manera que el efecto de la herencia por cruce no se vea diluido en exceso.

El fundamento de la mutación se encuentra en las características del proceso de búsqueda en el espacio de soluciones. Maley 1995 y Smith 1996 han realizado trabajos utilizando probabilidades de mutación adaptaba. Al comienzo de la evolución debe predominar la exploración del espacio por lo que la probabilidad de mutación debe ser alta, de manera que se realice una aproximación burda a la solución, para después disminuir la probabilidad de mutación cuando se trate de refinar la búsqueda sobre un espacio más restringido. Aunque otra vía sería la de incrementar la tasa por la mitad de la búsqueda para evitar una convergencia prematura.

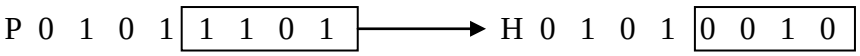
Mutación codificación binaria:

La mutación más simple es cambiar el valor del bit, un 0 por un 1 o al contrario.



Mutaciones sobre genes:

Son análogas al cruce externo en la medida en que afectan a un gen completo, no a un bit. Dado que un gen puede tomar más de dos valores, la mutación se realiza sustituyendo el valor actual por otro más o menos parecido.



Mutaciones no estacionarias:

El algoritmo genético debe realizar una búsqueda lo más amplia posible en las primeras etapas, mientras que en las últimas la búsqueda debería ser más local, por eso es conveniente ir reduciendo la tasa de mutación según progresa el algoritmo genético.

Un ejemplo sencillo sería ir reduciendo la probabilidad de mutación cada cierto periodo de tiempo.

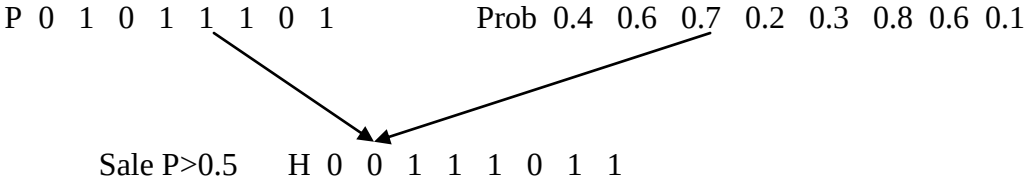
Si (número_generación mod $\beta = 0$) entonces
Probabilidad_mutación = Probabilidad_mutación – α

Ejemplo numérico

Si (numero_generación mod 10=0) entoces Pmutación=Pmutacion-0.09
Numeración generación =49 Probabilidad mutación = 0.65
Numeración generación =50 Probabilidad mutación = 0.56
.....
Numeración generación =59 Probabilidad mutación = 0.56
Numeración generación =60 Probabilidad mutación = 0.47
.....
Numeración generación =70 Probabilidad mutación = 0.38
.....
.....

Mutaciones no uniformes:

Consiste en dar distintas probabilidades de mutación a cada gen o cada bit dentro de un cromosoma, dependiendo de su significado.



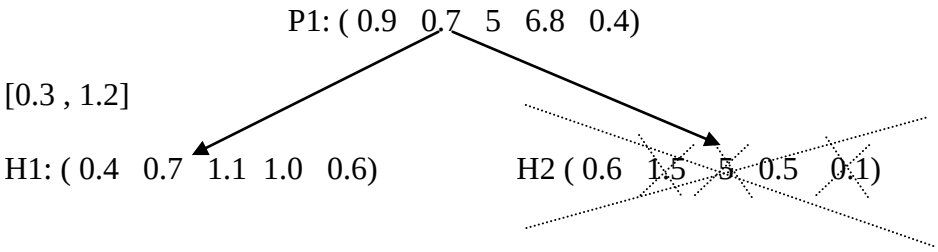
Mutación en codificación real

Existen diversos métodos de mutación en codificación reales. Se indica dos muy representativos.

Mutación Uniforme:

Este operador se aplica con una probabilidad dada sobre cada individuo. Cuando uno de los individuos es seleccionado para la mutación y todas sus componentes tomarán unos valores completamente aleatorios en el intervalo seleccionado.

En el ejemplo que se muestra el H2 no sería valido porque tiene valores que están fuera del rango indicado



Mutación No Uniforme:

Existe una variable binaria que se lanzará D veces, una por cada componente del vector. A cada componente de vector se le asignará un valor uniformemente distribuido dentro del intervalo, según la máscara. Así el operador actúa de forma más local que en la mutación uniforme, impidiendo saltos demasiado bruscos en los individuos. A veces estos desplazamientos aleatorios son efectuados según leyes gaussianas, en vez de uniformes.

Un ejemplo sencillo con la siguiente máscara de valores y valores uniformados es**:**

m: (0 1 0 1 1) 1->[0.0 ,0.8]; 0-> [0.5,1.0]

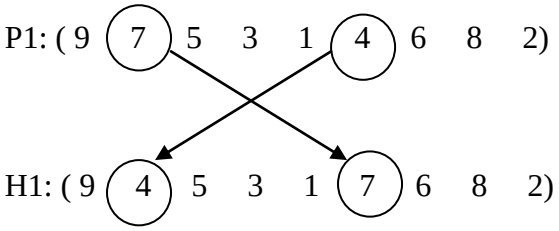
P1: (0.9 0.7 0.5 0.8 0.4)

H1: (0.7 0.3 0.9 0.2 0.4)

Mutación en codificación de enteros.

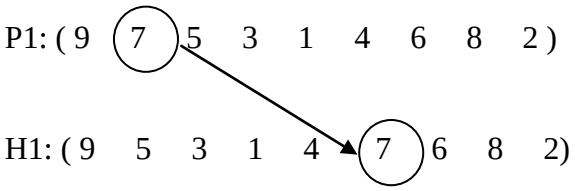
Mutación Intercambio Recíproco:

Al menos dos elementos se intercambian la posición de forma aleatoria.



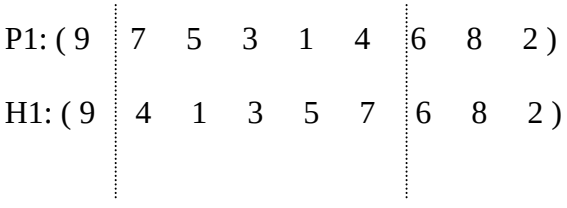
Mutación Inserción:

Se selecciona un elemento y se cambia de lugar de forma completamente aleatoria.



Mutación Inversión:

Se seleccionan dos puntos de listas y los valores que se encuentra entre esos puntos se intercambian entre ellos.



Mutación Barajada:

Es un método idéntico al anterior con la salvedad que los elementos que se encuentran entre los dos puntos seleccionados se cambian de forma aleatoria, sin ninguna regla fija.

P1: (9	7	5	3	1	4	6	8	2)
H1: (9	1	3	5	4	7	6	8	2)

La elección de un método u otro no sólo depende del tipo de problema que se desea resolver sino también debe tenerse en cuenta que no viole ninguno de los requisitos imprescindibles para el buen funcionamiento del algoritmo genético.

Reselección

Consiste en seleccionar para formar parte de la nueva generación a una combinación de los padres y los hijos generados, de tal forma que pasen, en cierta manera, los mejores entre ellos.

Lo que está claro es que al final de todos los procesos, la nueva población generada debe ser del mismo tamaño que la generación anterior.

Problemática del operador Reproducción

La selección o reproducción controla el carácter de la búsqueda y debe mantener un compromiso muy importante, ya que si se premian excesivamente a los cromosomas con alto fitness en las primeras generaciones se puede incurrir en el problema de convergencia prematura o sobreexplotación. Mientras si se bonifica insuficientemente a los individuos con alto fitness se estará contradiciendo los objetivos de la búsqueda de los óptimos.

Una consecuencia del mecanismo de selección proporcional es la obtención de individuo subóptimos con alto fitness relativo al resto de la población. Pudiendo este individuo dominar la población, llegando a una convergencia prematura.

Para solucionar estos problemas pueden utilizarse los siguientes métodos.

- Usar distribuciones estáticas para las probabilidades de selección en lugar de distribuciones dinámicas, asociada a la función de fitness de cada individuo. Una aproximación a esta técnica se llama orden lineal (Bäck 1991) y propone que, las probabilidades de selección serán valores constantes. Los resultados obtenidos han sido favorables, aunque han sido ignorados por bastante tiempo.

- Usar el escalado del fitness que consiste en transformar los valores mediante una aplicación. El objetivo de esta técnica es expandir el rango de los valores del fitness, de manera que se penalizan a los superindividuos.

Problemática del operador Mutación:

Una problemática de los algoritmos es la probabilidad de mutación. Por tanto, sería interesante encontrar alguna relación que indique qué valores escoger para ella en cada aplicación.

Una aproximación bastante prometedora es la utilización de una probabilidad de mutación adaptativa, es decir, que sea el propio algoritmo el que encuentre los valores óptimos como si formara parte del problema. Su adaptación a los AG consiste en:

- El genotipo de los individuos es ampliado para incluir la codificación de 1 o n (siendo n el número de variables que el cromosoma codifica) y la probabilidad de mutación.
- La probabilidad de mutación de cada individuo es descodificada para obtener un valor entre [0,1]
- Como la probabilidad de mutación pertenece a la cadena de individuo, cuando se muta el individuo puede que también la probabilidad sea modificada para siguientes generaciones.

Eliminación de individuos repetidos

La existencia de individuos idénticos disminuye el potencial de la población para la obtención de los resultados deseados. Siendo más favorable la diversidad de la población al cubrir más espacio de búsqueda.

- Remoción de Clones: Si tras la procreación, dos o más individuos resultan ser clones entre sí, uno de ellos será sometido a mutación hasta que ambos difieran de forma apreciable.
- Prevención de Incestos: Cuando se elige para el cruce una pareja de individuos idénticos se rechazan y se elige otro.

Criterios de paradas

No hay un límite fijo en el número de pasos que puede dar un algoritmo genético para encontrar la mejor solución posible, por lo que hay que fijar unos criterios de parada y una relación entre ellos. Hay muchas posibilidades, aquí se enumeran sólo algunos ejemplos de uso frecuente.

Número máximo de generaciones: El algoritmo se ejecuta durante un número de N de generaciones, tras el cual se detiene.

Valor de evaluación: El algoritmo se ejecuta hasta que se alcanza o se supera un valor de evaluación (fitness) determinado.

Número de generaciones sin mejora: Se guarda un registro de la última generación en la que hubo mejora en la evaluación, definido generalmente como el valor del mejor

individuo. Se permite que el algoritmo progrese un número N de generaciones sin que haya mejora.

Una medida de control sobre la evolución del algoritmo consiste en el cálculo de la desviación típica de los valores de evaluación en cada generación. Cuando ésta disminuye significa que se está perdiendo variedad genética y que el algoritmo está convergiendo.

Criterio de tratamiento de los individuos no factibles.

No siempre es posible establecer una correspondencia entre las soluciones de un problema y el conjunto de las cadenas binarias usadas para resolverlo, lo cual puede suponer un inconveniente a la hora de explorar el espacio de búsqueda. Se pueden emplear algunas técnicas para resolver este problema, cada una con sus ventajas y desventajas:

Técnicas de penalización: Resuelve directamente el problema, penalizando a los individuos no factibles. La única exigencia de esta técnica es la posibilidad de medir el "grado no-factible" para dar mayor penalización a los puntos más alejados a la solución.

Técnicas de reparación o corrección: Consiste en crear un procedimiento con el que corrija cualquier solución no factible que se genere. Un inconveniente de esta técnica es la de encontrar el procedimiento de corrección, el cual varía según el problema que se desea resolver.

Técnicas de decodificación: Realiza la codificación de tal manera que siempre genere soluciones factibles. El inconveniente es encontrar la codificación para cada problema.

Paradigma del recuerdo del comportamiento: Esta técnica se usa cuando el problema posea restricciones. Una manera de abordarlo consiste en encadenar dos AG con dos funciones de fitness diferentes. El primero utiliza como función de fitness aquella que mide exclusivamente el grado de satisfacción, de modo parecido a las funciones de gratificación. El segundo AG coge como población inicial la final del primer algoritmo y evoluciona el problema, devolviendo cero cuando no satisfagan las restricciones

Inclusión de restricciones.

A veces los problemas de optimización se plantean sujetos a restricciones. En algunos casos estas alteraciones pueden resultar muy inadecuadas para el progreso de un algoritmo de búsqueda hacia las soluciones válidas. Existen dos técnicas que ayudan a los AG en su tarea de optimización con restricciones:

- Penalización: La función de fitness estará formada por la suma de la función de adecuación más un término de penalización que castiga a los individuos que no cumple las restricciones del problema. Esta función de penalización puede depender de la generación en la que se encuentre el algoritmo, haciendo que al principio el algoritmo sea más flexible y según avance en las generaciones sea

más restrictiva. Esta solución es precaria porque si los valores de la función de fitness son pequeños, no será capaz de eliminar soluciones no válidas. Y si los valores son demasiados grandes, hará que el proceso se concentre en buscar soluciones validas, sin favorecer el sesgo hacia la mejor adecuación.

- Reparación: El método consiste en que los individuos recién generados que se encuentren fuera de las regiones permitidas sean reformados en nuevos individuos que si cumplen las restricciones. Es método puede llevar una complejidad considerable.

Lo más adecuado en estos casos es diseñar los algoritmos evolutivos ajustando sus componentes, ya sea parámetros, operadores o estructuras de datos, al problema que se desea optimizar.

3.7 Cómo funciona y por qué funciona los Algoritmos Genéticos.

El mecanismo de un AG es simple y la explicación de por qué funciona es bastante más compleja. No existe una teoría general aceptada para explicar el funcionamiento del AG. La simplicidad y la potencia de estos algoritmos son sus principales puntos fuertes.

Existe diferentes hipótesis que intentan explicar él por que funcionan. A continuación se enumeraran dos de estas hipótesis: la primera presentada por John Holland es el teorema de esquema y la otra por David E. Goldberg hipótesis de los bloques básicos.

Teorema de esquema:

Es una de las primeras explicaciones rigurosa que explica el funcionamiento del AG, proporcionada por (Holland 1975).

Para explicar cómo y por qué funciona un algoritmo genético es necesario introducir un concepto esquema, que consiste en representar el grado de parecido o similitud entre los individuos de una misma población. Un esquema es una estructura de valores de genes que pueden representarse (en codificación binaria) como una cadena de caracteres en el alfabeto {0,1,#}. Se emplea el signo comodín "#" puede corresponder a cualquier valor 0 o 1. Un ejemplo para una población de individuos que posee cadenas binarias de longitud 3, si el esquema fuese "###" representaría a todos los individuos de la población. Mientras si el esquema es "10#" solamente podría representar al individuo "100" o el individuo "101".

Lo que procesa realmente un algoritmo genético son los esquemas. De esta manera al evolucionar la población inicial, generación tras generación, los esquemas más aptos irán sobreviviendo mientras que los esquemas menos aptos irán desapareciendo. Si se habla de esquema es importante saber cuál es el orden de un esquema o(S) y cuál es su longitud característica l(S), ya que dependiendo de ambos parámetros se puede predecir qué esquemas son los más aptos.

El orden de un esquema es el número de bits distintos de # dentro de una cadena binara. Un ejemplo ##10#0# tiene orden 3. La longitud característica de un esquema es la máxima distancia medida en número de bits entre las posiciones fijas (bits a 1 ó 0) de los dos extremos, es decir, el número cuyos bits coinciden con #. En el esquema anterior ##10#0# la distancia característica es 4.

El teorema de los esquemas explica que la potencia del AG es cómo son procesados los esquemas. En la selección, a los individuos de una población posee la oportunidad de reproducirse, generando descendencia para la siguiente generación. El número de oportunidades que recibe un individuo es proporcional a la función fitness, de esta forma que los mejores individuos contribuyen con sus genes de manera importante a la construcción de generaciones posteriores. Con cada reproducción se están propagando de generación en generación los mejores esquemas que aumentarán la probabilidad de encontrar las soluciones mejores.

Holland explicó que el camino óptimo para explorar el espacio de búsqueda es dar a cada individuo una probabilidad de continuar en la siguiente generación, en proporción a su nivel de fitness relativo. De esta forma, los esquemas que aportan calidad a un individuo, reciben probabilidades exponencialmente creciente en generaciones sucesivas.

En los algoritmos genéticos se puede deducir matemáticamente que los esquemas más aptos son aquellos que presentan bajo orden y corta longitud característica, ya que al aplicar sobre ellos los operadores de cruce y mutación es más difícil eliminarlos.

Hipótesis de los bloques básicos

El teorema fundamental permite formular una serie de ideas acerca de por qué los AG funcionan. Los bloques básicos o bloque constructivos son esquemas muy adaptados de baja longitud característica y bajo orden. Del apartado anterior se concluye que los esquemas altamente adaptados son de gran importancia en el proceso genético, ya que son los que van a dominar a la población llevándolo hacia regiones de alta calidad.

La fuerza de los AG está en la posibilidad de combinar los bloques básicos para forma cada vez mejores individuos. Pero quien tiene realmente la responsabilidad de crear dichos bloques es la representación seleccionada. Si en una codificación de un AG no existen dichos bloques básicos, el proceso no será dirigido y por lo tanto no se podrá asegurar un resultado óptimo.

Paralelismo implícito

Un cromosoma representa a 2^k esquemas diferentes cuando su longitud es k. Por lo tanto se puede establecer un límite superior y otro inferior para contabilizar el número de esquemas totales que son procesados en una población formada por *n* individuos de longitud *k.* Para el límite superior se supondrá que todos los cromosomas de la población son distintos y que además los esquema procesados por cada individuos son diferentes en su conjunto. Entonces el límite superior será $n * 2^k$. El límite inferior es

que todos los individuos son iguales por tanto el número mínimo de esquemas procesados es de 2^k. El AG opera implícitamente sobre un número de esquema mucha mayor que el tamaño de la población.

La estrategia de búsqueda en AG´s incrementa exponencialmente a los individuos con mejor fitness, pero sin dejar de probar nuevas soluciones. El crecimiento de los individuos es exponencial en aquellos que se estiman como mejores frente a aquellos que se estiman peores.

3.8 Ventajas y desventajas de los Algoritmos Genéticos.

Algunas de las ventajas que presentan los Algoritmos Genéticos respecto a otras técnicas de optimización son:

- Los AG no requieren que la función objetivo tenga un "buen comportamiento", ya que son capaces de tolerar sin ningún problema discontinuidades o funciones poli-nominales.
- Los AG constituyen el tipo de Técnicas Evolutivas más completo porque reúne de un modo intuitivo toda la base de Técnicas Evolutivo.
- De todos los procedimientos relacionados con las Técnicas Evolutiva son los que menos conocimiento específico necesita sobre el tema.
- Los AG se prestan para implementaciones distribuidas o en paralelo.
- Debido a que los AG realizan una búsqueda en una población de puntos y están basados en reglas de transición probabilísticas identificando un mayor número de máximos, es menos factible que converjan a un máximo o mínimo local.
- Los AG pueden incorporar conocimiento adicional del problema de un modo muy sencillo.
- No son necesarios que los programas se ejecuten en ordenadores con mucha capacidad, ya que no necesitan de grandes cálculos.
- Son simples de implementar, ya que la única información necesaria es la función objetivo y las restricciones correspondientes.
- Posee muchas posibilidades de modificación y adaptación al problema a resolver de una forma muy sencilla.
- A los algoritmos genéticos se les considera métodos de búsqueda global que no emplean la información del gradiente, por lo que son apropiados para optimizar funciones no diferenciables o funciones con profusión de mínimos locales.
- Los AG esta guiado por la aptitud de los individuos (fitness) y no incorpora ningún otro conocimiento específico del problema en cuestión.

Las desventajas de los AG son:

- El mecanismo del algoritmo no considera las posibles restricciones o ligaduras que pudieran imponerse a la búsqueda.
- El Algoritmo genético sólo puede trabajar con poblaciones finitas y el espacio de búsqueda para que funcione correctamente debe ser acotado.

- Convergencia prematura, Pérdida de variabilidad genética impide producir soluciones novedosas. Se pierde la capacidad de intercambio de información útil entro los individuos. Se produce la pérdida de variabilidad en las primeras fases pudiéndose estancar en algún mínimo local.
- El comportamiento no determinista de la selección respecto del valor de adaptación del efecto de la mutación y de la movilidad del valor de adaptación de los individuos. Se ha demostrado que a menos que se use elitismo, el AG no puede converger a la solución óptima, aunque sí puede encontrarla. El elitismo puede ocasionar por otra parte una convergencia prematura, lo que no es deseable, sobre todo si los problemas son engañosos, pudiendo caer en dicho error.
- Problemas de Epítasis: Los AG se fundamente en la adyacencia. Si no se cumple esta propiedad, los nuevos individuos se generan más aleatoriamente entonces el proceso de búsqueda es menos heurístico.
- Cuando la función de evaluación es muy abrupta, no existe suficiente presión selectiva para aproximarse a la solución óptima. Todos los individuos tienen la misma probabilidad de converger en la solución. Convirtiéndose en un proceso aleatorio
- Si la información proporcionada resulta insuficiente para orientar correctamente al algoritmo en su búsqueda del óptimo, se produce una desorientación. Puede ocurrir que la contribución a la aptitud de un individuo que realiza cierto gen depende de los valores que tomen otros. A este fenómeno se le llama epítasis o acoplamiento y su presencia garantiza la desorientación.
- Los algoritmos genéticos, frente a otros métodos de optimización, son un compromiso entre una búsqueda fuerte y una búsqueda débil. La búsqueda es mucho más fuerte que en el caso aleatorio, pero a la vez demasiado débil como para poder demostrar que se ha alcanzado un óptimo global. Este tipo de métodos se conoce como métodos débiles y son métodos robustos y generales, porque trabajan con muy poca información previa sobre el problema.
- Siguiendo la exposición hay dos factores esenciales para determinar la dificultad de un problema de optimización: la representación y el método de optimización.
- Una dificultad añadida sería cómo se debe plasmar los problemas de forma indicativa, unívoca y no ambigua en forma de genes, es decir encontrar a veces una buena representación de las soluciones es complicado.

3.9 Comparación con otros métodos de búsqueda

En la actualidad existen tres tipos principales de algoritmos de búsqueda, que son:

- Métodos basado en el cálculo diferencial (calculus-based methods):

Existen dos grupos principales, métodos directos e indirectos.

Los métodos indirectos buscan los extremos locales resolviendo las ecuaciones, realizando el gradiente de la función objetivo igualándola a cero. Esto es una generalización multidimensional del concepto elemental del cálculo de los extremos. Es decir, dada una función se buscan posibles picos comenzando la

búsqueda con una restricción tal que serán solución aquellos puntos con pendiente igual a cero en todas las direcciones.

Los métodos directos buscan óptimos locales mediante saltos en la función y movimientos en la dirección relacionada al gradiente local, es simplemente la idea de escalar una montaña para encontrar el óptimo local.

Estos métodos poseen menor robustez que los Algoritmos Genéticos porque son métodos (directos e indirectos) que tienen alcance local, es decir, busca el mejor en una cercanía. Además, depende que la función sea diferenciable en todos sus puntos, es decir, que posean derivadas y la función sea continua en todos sus puntos. Por estas razones se pueden indicar que estos métodos poseen menor robustez que los Algoritmos Genéticos.

- Esquemas enumerativos (enumerative schemes).

Dentro de un espacio finito de búsqueda o en un espacio infinito discreto, el algoritmo de búsqueda comienza buscando los valores de una función objetivo en cada punto del espacio cada vez. Este tipo de algoritmos es bastante simple y es una forma humana de búsqueda, y puede resultar útil cuando el número de posibilidades es pequeño. Posee la desventaja en cuanto a la robustez. Si los espacio de búsqueda son demasiado grandes, suelen ser poco eficientes.

- Algoritmos aleatorios de búsqueda (random search algortihms) :

Son algoritmos que usa algún grado de aleatoriedad como para de su lógica. Utiliza normalmente al azar de manera uniforme bits como una entrada auxiliar para guiar su comportamiento, con la esperanza de logra un buen desempeño en el promedio de los casos.

Los algoritmos genéticos usan un procedimiento de búsqueda que usa selección aleatoria como herramienta, codificando un espacio parámetros y la búsqueda de soluciones idóneas a un problema consiste en la búsqueda de determinadas cadenas binarias. El universo de todas las posibles cadenas puede ser correspondido a soluciones menos buenas Además, la búsqueda de los AG's es mucho más fuerte que la búsqueda aleatoria, al usar parte de la información para poder llegar a los óptimos deseados.

3.10 Aplicaciones de los Algoritmos Genéticos

Las aplicaciones de los Algoritmos Genéticos son numerosas y muy variadas. Como se ha se ha comentado, los AG sirven para resolver problemas de optimización, ya que estos problemas fueron la fuente de inspiración de sus creadores. Se han utilizado en numerosas tareas de optimización, incluyendo la optimización numérica, y los problemas de optimización combinatoria. Son utilizados en diversos problemas y modelos de ingeniería, y de la ciencia en general. De manera general, cabe destacar entre ellos los siguientes:

- Programación automática: Los Algoritmos Genéticos se han empleado para desarrollar programas para tareas específicas, y para diseñar otras estructuras computacionales tales como el autómata celular, y las redes de clasificación.

- Aprendizaje máquina: Los algoritmos genéticos se han utilizado también en muchas de estas aplicaciones, tales como la predicción del tiempo o la estructura de una proteína. Han servido asimismo para desarrollar determinados aspectos de sistemas particulares de aprendizaje, como puede ser el de los pesos en una red neuronal, las reglas para sistemas de clasificación de aprendizaje o sistemas de producción simbólica, y los sensores para la robótica.

- Economía: En este caso, se ha hecho uso de estos algoritmos para moldear procesos de innovación, desarrollo de estrategias de puja, y la aparición de mercados económicos.

- Sistemas inmunes: A la hora de crear varios aspectos de los sistemas inmunes naturales, incluyendo la mutación somática durante la vida de un individuo y el descubrimiento de familias de genes múltiples en tiempo evolutivo, ha resultado útil el empleo de esta técnica.

- Ecología: En los fenómenos ecológicos tales como las carreras de armamento biológico, la co-evolución de parásito-huésped, las simbiosis y el flujo de recursos.

- Sistemas sociales: En el estudio de aspectos evolutivos de los sistemas sociales, tales como la evolución del comportamiento social en colonias de insectos, y la evolución de la cooperación y la comunicación en sistemas multi-agentes.

Gracias al éxito en estas aplicaciones y otras áreas, los Algoritmos Genéticos han llegado a ser un campo puntero en la investigación actual, habiendo alcanzado un gran desarrollo e importancia.

3.11 Utilidades y aplicaciones de los AG.

A continuación se mostrarán algunas aplicaciones en las que se han utilizado los AG. Se demostrará que el poder de los algoritmos genéticos gana reconocimiento cada vez más, ya que son utilizados para abordar una amplia variedad de problemas en un conjunto de campos sumamente diverso.

Acústica.

En (Sato et al. 2002) utilizaron algoritmos genéticos para diseñar una sala de concierto con propiedades acústicas óptimas, maximizando la calidad del sonido para la audiencia, para el director y para los músicos del escenario. (Tang et al.,1996) analiza el uso de los algoritmos genéticos en el campo de la acústica y el procesamiento de señales. Un área de interés particular incluye el uso de AG para diseñar sistema de Control Activo de Ruido (CAR) que elimina el sonido no deseado produciendo ondas sonoras que interfieren destructivamente con el ruido.

Ingeniería aeroespacial.

(Obayashi et al.2000) utilizaron un algoritmo genético de múltiples objetivos para diseñar la forma del ala de un avión supersónico. Hay tres consideraciones principales que determina la configuración del ala, minimizar la resistencia aerodinámica a velocidades, minimizar la resistencia a velocidad y minimizar la carga aerodinámica. En (Sasaki et al. 2001) los autores repitieron el experimento añadiendo un cuarto objetivo, que consistía en minimizar el momento de torsión (un problema de los diseños de alas flecha en el vuelo supersónico).

(Williams, Crossley y Lang 2001) aplicaron algoritmos genéticos a la tarea de situar órbitas de satélites para minimizar los apagones de cobertura. Cuando se utilizó el AG en este problema, los resultados que evolucionaron para constelaciones de tres, cuatro y cinco satélites eran extraños, configuraciones orbitales muy asimétricas, con los satélites colocados alternando huecos grandes y pequeños, en lugar de huecos de igual tamaño como habrían hecho las técnicas convencionales. Sin embargo, esta solución redujo significativamente los tiempos medio y máximo de apagón, en algunos casos hasta en 90 minutos. En un artículo periodístico, el Crossley señaló que ``ingenieros con años de experiencia aeroespacial quedaron sorprendidos con el rendimiento ofrecido por el diseño no convencional"

(Keane y Brown 1996)) Utilizaron un AG para producir un nuevo diseño para un brazo o jirafa para transportar carga que pudiese montarse en órbita y utilizarse con satélites, estaciones espaciales y otros proyectos de construcción aeroespacial. El resultado, una estructura retorcida con aspecto orgánico que se ha comparado con un fémur humano, no utiliza más material que el diseño de brazo estándar, pero es ligera, fuerte y muy superior a la hora de amortiguar las vibraciones perjudiciales

También Gibas 1996 ha utilizado un algoritmo genético para producir mediante evolución una serie de maniobras para mover una nave espacial de una orientación a otra. La solución era un 10 % más rápida que la solución producida manualmente por un experto

Astronomía y astrofísica.

(Charbonneau 1995) sugiere la utilidad de los AG para problemas de astrofísica, aplicándolos a tres problemas de ejemplo: obtener la curva rotación de una galaxia basándose en las velocidades rotacionales de sus componentes, determinar el periodo de pulsación de las estrellas y sacar los valores de los parámetros críticos de un modelo magneto-hidrodinámico del viento solar. Son problemas difíciles de resolver, no lineales y multidimensionales. Pare el primer problema, la rotación galáctica, el AG produjo dos curvas, y ambas estaban bien ajustadas a los datos. En el segundo problema, las pulsaciones de una estrella, se basaron en una serie temporal, y tuvieron unos resultados de gran calidad. Y en el último problema, determinar los seis parámetros críticos del viento solar, con valor de 3 obtuvieron una precisión de menos del 0,1% y las otras tres entre 1 y 10 %. Charbonneau indicó que no existe ningún método más eficiente y robusto para resolver experimentalmente un problema no lineal 6-dimensional.

Química.

En un artículo de (Gillet 2002), se describe el uso de un algoritmo genético multiobjetivo para el diseño basado en los productos de bibliotecas combinatorias. Al elegir los componentes que van en una biblioteca particular, deben considerarse características como la diversidad y peso molecular, el coste de los suministros, la toxicidad, la absorción, la distribución y el metabolismo. Si el objetivo es encontrar moléculas similares a una molécula existente con una función conocida (un método común en el diseño de nuevos fármacos), también se puede tener en cuenta la similitud estructural. En ese artículo se presentó un enfoque multiobjetivo donde puede desarrollarse un conjunto de resultados que maximicen o minimicen cada de estos objetivos.

En un artículo creado por (Glen y Payne 1995) se describe el uso de algoritmos genéticos para diseñar automáticamente moléculas nuevas desde cero, que se ajustan a un conjunto de especificaciones dado.

Ingeniería eléctrica.

(Altshuler y Linden 1997) utilizaron un algoritmo genético para evolucionar antenas de alambre con ciertas propiedades. Los autores indicaron que el diseño de las antenas es un proceso impreciso, intuitivo, ecuaciones aproximadas o estudios empíricos, pero al utilizar AG concluyeron que estas técnicas se muestran "excepcionalmente prometedoras y son capaces de resolver de manera efectiva difíciles problemas de antenas y será especialmente útil en situaciones en las que los diseños existentes no sean adecuados".

Mercados financieros

(Mahfou y Mani 1996) utilizaron algoritmos genéticos para predecir el rendimiento futuro de las acciones ofertadas públicamente. Al AG se le asignó la tarea de predecir el beneficio relativo de cada acción, definido como el beneficio de esa acción menos el beneficio medio de las acciones a lo largo del periodo de tiempo en cuestión. Al AG se le proporcionaros datos históricos de cada acción en forma de una lista de atributos, como la relación precio-beneficio, el ritmo de crecimiento; se le pide al AG que evolucionara un conjunto de reglas si/entones para clasificar cada acción y proporcionar, como salida, una recomendación sobre qué hacer con respecto a la acción (si comprar o vender). Aunque el mercado de valores es un sistema extremadamente ruidoso y no lineal, y ningún mecanismo predicativo puede ser correcto el 100% , el reto consistía en encontrar soluciones más precisas la mitad de las veces.

La utilización de los AG en los mercados financieros han empezado a extenderse en las empresas de corretaje bursátil del mundo real. (Naik 1996) informa de que LBS Capital Management, una empresa estadounidense con sede en Florida, utiliza algoritmos genéticos para escoger las acciones de los fondo de pensiones que administra. (Coale 1997) y (Begley y Beals1995) informan de que First Cuadrante, una empresa de inversiones de California que mueve 2.2000 millones, utiliza AG para la toma decisiones de inversión en todos sus servicios financieros. El modelo gana de media 225 dólares por cada 100 dólares invertidos durante seis años, en contraste con los 205 dólares cuando son utilizados otros modelos.

Juegos.

(Chellapilla y Fogel 2001) utilizaron los AG para evolucionar redes neuronales para que pudieran jugar a las damas. Una de las dificultades para este tipo de problema está relacionada con estrategias. Se pensaba que con unos criterios simples de ganar, perder o empatar no proporcionan la suficiente información para que el AG. Pero Chelapile y Fogel indican que con las posiciones especiales de las piezas y el número de piezas, fueron capaces de evolucionar el programa para crear jugadas a un nivel con expertos humanos. El algoritmo representaba una lista numérica de 32 elementos, en donde cada posición de la lista correspondía a una posición del tablero. El valor puede ser 0 desocupado, -1 ocupada por una pieza enemiga, +1 ocupado por una pieza amiga y –K y +K ocupada por una dama enemiga o amiga. Uno de los mejores individuos creados se inscribió en una página web de juegos, y durante un periodo de tiempo compitió con oponentes humanos de un alto nivel. De 165 oponentes ganó 94, perdió 29 y quedaron en tablas con 32.

Geofísica.

(Sambrige y Galleguer 1993) utilizaron un algoritmo genético para localizar los hipocentros de los terremotos, es decir el punto donde se origina un terremoto, basándose en datos sismológicos. Es un proceso bastante complicado, ya que las propiedades de las ondas sísmicas dependen de las capas de las rocas a través de las que viajan. Los métodos tradicionales para localizar el hipocentro se basa en el algoritmo de inversión sísmico. Se inicia con la mejor estimación de la ubicación, luego se calcula las derivadas de tiempo de viaje de la onda respecto al punto de origen, y realiza una operación para proporcionar una ubicación actualizada. Este proceso se repetirá hasta alcanzar una solución aceptable. Como todos lo AG comienza con una nube de hipocentros potenciales que encoge con el tiempo hasta converger en una solución. Los autores concluyen que su algoritmo genético es una herramienta nueva y poderosa para realizar una búsqueda robusta de hipocentros.

Ingeniería de materiales.

(Giro, Cyrillo y Galvao ,2002) utilizan algoritmos genéticos para diseñar polímeros conductores electricidad basados en el carbono. Se trata de un material sintético creado recientemente, posee grandes aplicaciones tecnológicas potenciales, aunque debido a su alta reactividad, los átomos de carbono pueden formar un número virtual infinito de estructuras. Los autores aplican un enfoque basado en AG a la tarea de diseñar moléculas nuevas con propiedades específicas a priori. Es una herramienta muy efectiva para guiar a los investigadores en la búsqueda de nuevos compuestos y extenderse al diseño de nuevo materiales que pertenezcan virtualmente a cualquier tipo de moléculas.

(Weismann, Hammel y Bäck 1998) aplicaron algoritmos evolutivos a un problema industrial de una alta dificultad, como es el diseño de revestimientos ópticos multicapa para filtros que reflejan, trasmiten o absorben luz de frecuencias especificadas. Ejemplo, la creación de gafas de sol.

También se debe nombrar la utilización de los AG para diseñar patrones de exposiciones para un haz de electrones de litografía utilizado para grabar estructuras a una escala menor del micrómetro en un circuito integrado, estudiado por Robin 2003 El diseño de esos patrones es una tarea muy difícil, bastante pesado y costoso, ya que se tienen que optimizar casi simultáneamente hasta 100 parámetros para controlar el haz de electrones.

Matemáticas y algoritmia.

(Haupt y Haupt 1998) describen el uso de los AG´s para la resolución de ecuaciones de derivadas parciales no lineales de alto orden. También para ordenar una lista de elementos, lo cual es un tema muy importante en la informática.

(Koza et al1999) utilizó programación genética para evolucionar redes de ordenación mínimas para conjuntos de 7 elementos (16 comparaciones), 8 elementos (19 comparaciones), y conjuntos de 9 con 25 comparaciones. Mitchell 1996 describe el uso de algoritmos genéticos para encontrar una red de ordenación de 61 comparaciones para un conjunto 16 elementos.

Para finalizar este punto, se debe mencionar que en el campo de la algoritmia (Koza en 1999) se utilizó programación genética para descubrir una regla para el problema de clasificación por mayoría en autómata celulares de una dimensión, una regla mejor que todas las reglas creadas por humanas.

Cumplimiento de órdenes o leyes.

(Kweley y Embrecths 2002) utilizaron algoritmos genéticos para evolucionar planes tácticos para las batallas militares. Se trata de una tarea compleja multidimensional que a menudo atormenta a los profesionales, no sólo porque este tipo decisiones crea un estrés sino que además intervienen un gran número de variables, como bajas amigas, bajas enemigos, el terreno, el tiempo meteorológico, recurso humanos, recursos tecnológicos etc. Desarrollaron un algoritmo genético que genere automáticamente la creación de planes de batalla, en conjunción de un simulador de batalla. Se realizaron diferentes pruebas con unos grupos de expertos militares y se obtuvo un rendimiento medio significativamente mayor al de los generados por los expertos militares. Además se obtuvieron unos resultados tácticos.

(Naik 1996) usó los AG´s en el cumplimiento de la ley, describiendo el software "FacePrints". Se trata de un proyecto para ayudar a los testigos a identificar y describir a los sospechosos criminales. La mayoría de las personas no son buenas describiendo aspectos individuales del rostro de una persona como el tamaño de la nariz, mandíbula, etc. Pero sí es más fácil reconocer una cara completa. El programa muestra a los testigos imágenes de rostros generadas aleatoriamente y el testigo elegirá aquellas caras que se parecen más al sospechoso. Y dichas caras se mutarán y combinaran para obtener el retrato del sospechoso.

Biología molecular.

(Koza et al. 1999) utiliza la programación genética para identificar el dominio transmembrana de una proteína. Las transmembrana son proteínas que sobresalen de una membrana celular y realizan a menudo funciones como detectar la presencia de algunas sustancias en el exterior de la célula o trasportarlas hacia el interior de la célula. Al algoritmo genético se le suministra operadores matemáticos estándares con los que trabaja, además de funciones booleanas para la detención de aminoácidos. Devuelve +1 si en una posición es detectada con el aminoácido correcto o –1 en caso contrario. Con esta sencillez, las aptitudes de las soluciones producidas por la programación genética fueron evaluadas probándolas con 246 segmentos proteínicos de los que se conocía su condición transmembrana. Luego se evaluó al mejor individuo de la prueba con 250 casos inéditos y se comparó su efectividad con la de los cuatro mejores algoritmos humanos creados con el mimo propósito. El resultado final produjo un algoritmo de identificación de segmentos transmembrana con una tasa total de error del 1.6% significativamente menor de los cuatro algoritmos humanos, el mejor de ellos tenía una tasa de error de 2.5%.

Reconocimiento de patrones y explotación de datos.

El mundo de las telecomunicaciones es feroz, porque existe una tasa de cambio de proveedor muy alta. La fuga de usuarios le cuesta a la compañía una gran cantidad de dinero cada año. Si las compañías pudieran contactar con los clientes que tienen probabilidad de cambiar y ofrecerles incentivos especiales para que no cambie de compañía, las pérdidas serían menores. En conclusión, el problema es intentar identificar a los clientes con intenciones de cambiar de compañía. Entonces (Au, Chan y Yao 2003) aplicaron los algoritmos genéticos a este problema para generar un conjunto de reglas de tipo si-entonces para predecir la probabilidad de que abandone la empresa los distintos grupos de cliente.

(Rizki, Zmunda y Tamburino 2002) utilizaron AG para evolucionar un complejo sistema de reconocimiento de patrones con una amplia variedad de usos potenciales. Realizaba el reconocimiento de patrones, detectores individuales de característica en forma de árboles de expresiones y luego se evolucionaban combinaciones cooperativas de esos detectores para producir un sistema completo de reconocimiento de patrones.

(Hughes y Leyland 2000) también aplicaron AG´s multiobjetivo a la tarea de clasificar objetivos basándose en sus reflexiones radar. Los datos de una sección transversal radar de alta resolución necesitan enormes cantidades de espacio de almacenamiento en disco, y producir un modelo realista de la fuente a partir de los datos es muy costoso computacionalmente. En contraste, el método basado en el AG demostró ser muy exitoso, produciendo un modelo tan bueno como el del método iterativo tradicional, pero reduciendo el gasto computancional y las necesidades de almacenamiento hasta el punto de que era factible generar buenos modelos en un ordenador personal.

Robótica

(Andre y Teller 1999) inscribieron a un equipo llamado Darwin United en el torneo internacional RoboCup, cuyos programas de control han sido desarrollados automáticamente desde cero mediante programación genética. Indicar que el torneo internacional RoboCup consiste en un campeonato de fútbol entre equipos de robots autónomos. Andre y Teller proporcionaron al programa genético un conjunto de funciones primitivas de control, como girar, moverse, tirar, etc... También se crea una lista de objetivos como acercarse a la pelota, golpear la pelota, conservar la pelota, moverse dirección correcta, marcar goles, etc... Los individuos se evalúan haciéndoles jugar contra un equipo estacionario, que no se mueve, pero golpea la pelota en sentido contrario. También se les hace jugar un partido en contra el equipo ganador de la competición RoboCup de 1997 y con los equipos que marcaron al menos un gol contra el equipo ganador. Los equipos que pasen, jugarán entre ellos para encontrar el mejor. De los 34 equipos Darwin United acabó en decimoséptimo posición.

Diseño de rutas y horarios.

(Burke y Newall 1999) utilizaron algoritmos genéticos para diseñar los horarios de los exámenes universitarios. El problema de los horarios es NP-completo, no conociendo un método para hallar con garantías una solución óptima en un tiempo razonable. El problema posee unas restricciones duras, que no coincida dos exámenes en la misma hora y misma clase, y restricciones suaves, es decir si es posible, intentar que no coincida dos exámenes en sucesión a un mismo estudiante. Los resultados obtenidos en el programa fueron comparados con otros programas que utilizan varias universidades, obteniendo una mejora del 40%.

(He y Mort 2000) aplicaron algoritmos genéticos al problema hallar rutas óptimas en rede de telecomunicaciones (redes telefonía e Internet) para transmitir datos desde los remitentes hasta el destinatario. Es un problema NP-completo, teniendo que equilibrar objetivos en conflicto, como maximizar el caudal de datos, minimizar los retrasos de transmisión y la pérdida de datos, encontrar camino debajo de coste y distribuir la carga uniformemente entre los conmutadores de la red. Cuando se probó sobre un conjunto de datos derivados de una base de datos en red real de Oracle se descubrió que el AG era capaz de redimir enlaces rotos o congestionados, equilibrar la carga de tráfico y maximizar el caudal total de la red. Esta técnica ha encontrado aplicaciones reales en la compañía de telecomunicaciones U.S West, que se enfrentó a la tarea de desplegar una red de fibra óptica.

Jensen, 2003 y Chryssolourios y Subramania, 2001 aplicaron algoritmos genéticos a la tarea de generar programas para líneas de montaje. Deben tener en cuenta el coste, retrasos, rendimiento y diversas complicaciones. Los AG´s producirán programas eficientes que pueden tratar con más facilidad los retrasos y las averías.

Ingeniería de sistemas.

Benini y Toffolo ,2002 utilizaron los algoritmos genéticos para la tarea de diseñar molinos eólicos para generar energía eléctrica. En la actualidad existen diferentes tipos de turbina y no hay acuerdo cuál es la óptima. Se busca obtener el

mayor rendimiento a un coste mínimo. El AG encontró diseños competitivos con los que existen en el mercado y a un precio aceptable.

Haas, Hurnham y Mills ,1997 usaron los algoritmos genéticos para optimizar la orientación e intensidad del haz de los emisores de rayos X utilizado en radioterapia, eliminado las células cancerosas y evitando el tejido sano. Utilizaron un método convencional, y luego utilizaron el AG para modificarlo y mejorarlo.

Lee y Zak, 2002 utilizaron un algoritmo genético para evolucionar un conjunto de reglas para controlar un sistema de frenos antibloqueo automovilístico, intentando reducir la distancia de frenada y mejorar la maniobrabilidad. El rendimiento del ABS depende de las condiciones de la carretera, de la meteorología, etc.. Proponen un AG para ajustar un controlador ABS que pueda identificar las propiedades de la superficie y que actúe en consecuencia, liberando la cantidad adecuada de fuerza de frenado para maximizar la tracción de las ruedas. En las pruebas se obtuvieron resultados bastante buenos, recomendado la utilización de AG para la mejor configuración de controladores

Schechter, 2000 indicó que el Dr. Meter Senecal de la Universidad de Wisconsin utilizó algoritmos genéticos de población pequeña para mejorar la eficiencia de los motores diésel. Estos motores funcionan inyectando el combustible en una cámara que está llena de aire comprimido y bastante caliente para hacer que la combustible explote y empuje un pistón. El método de Senecal produjo un motor mejorado que consumía un 15% menos de combustible y producía dos tercios menos de óxido nítrico de escape y mitad de hollín.

Texas Instrumens utilizó un algoritmo genético para optimizar la disposición de un chip informático. El AG con una estrategia de conexiones muy novedosa alcanzó un diseño el chip que ocupaba un 18% menos de espacio.

Empresas de la industria aeroespacial, automovilística, turbo-maquinaría y electrónica están utilizado un sistema de software propietario conocido como Engineous, que utiliza algoritmos genéticos para diseñar y mejorar motores, turbinas, y otros dispositivos industriales. Supuestamente Enginueos cuenta con algoritmos genéticos, también emplea técnicas de optimización numérica y sistemas expertos que utilizan reglas si – entonces, para imitar el proceso de decisión humana. Exploraron regiones del espacio de búsqueda que pasan por alto otros métodos. Enginueos se ha utilizado en la industria para abordar diferentes problemas. El más famoso es la utilización para mejorar la turbina generadora de energía del avión Boeing 777, consiguió una eficiencia en combustible del 1%. También Engineuos fue utilizado para optimizar la configuración de motores eléctricos industriales, generadores hidroeléctricos y turbinas de vapor, para proyectar redes eléctricas, para generar superconductores y generadores de energía nuclear para satélites en órbita. También la NASA utilizó Engineous para optimizar el diseño de un avión de gran altitud para analizar la disminución del ozono.

Capítulo 4. Algoritmos Genéticos de Nichos

4.1 Introducción

La modalidad de nichos facilita la capacidad de encontrar el óptimo global de una función, además de un conjunto de óptimo locales. Hasta ahora el resto de los Algoritmos Genéticos no eran capaz de realizarlo, solamente localizaba un óptimo de la función.

Los métodos de nichos son técnicas que promueven la formación y manutención de sub-poblaciones estables en el AG. Los métodos se pueden aplicar en la formación y manutención de sub-poblaciones intermedias orientadas a la obtención de una solución final única, aunque se suele utilizar en el contexto de formar y mantener múltiples soluciones finales.

Un método de nichos debe ser capaz de formar y mantener múltiples y diversas soluciones finales, aunque posean iguales o diferentes fitness. Además, debe ser capaz de mantener estas soluciones durante un periodo infinito de tiempo con respecto al tamaño de la población.

Con respecto al espacio de búsqueda, la convergencia puede ocurrir en algunas regiones locales, pero la diversidad debe orientar esta búsqueda a las regiones más prominentes o adaptadas.

Dado un conjunto de múltiples óptimos y una capacidad limitada para localizar un óptimo, el mejor AG con nichos deberá preferir el óptimo más alto. Algunos AG pueden preferir también algún óptimo que se encuentre en otro máximo parcialmente óptimo. El mejor método de nichos debe ser capaz de localizar los puntos más altos ante la presencia de un gran número de puntos inferiores, aun cuando se trate de problemas altamente engañosos. Más aún, el mejor método de nichos no siempre será altamente selectivo, sino también tendrá la habilidad de formar y mantener óptimos globales y no globales. Se centrará la atención en este análisis en la manutención en vez de formación de soluciones óptimas, ya que se asume que un AG simple es capaz de formar buenas soluciones, y esta capacidad por supuesto, será transmitida a un AG con nichos. Si esta capacidad no es transferible a un algún algoritmo en particular, se considerará que este algoritmo no corresponde a un método de nichos.

4.2 Marco formal para Escenarios Multimodales

Los escenarios multimodales constituyen el terreno donde los métodos de nichos van a ejecutarse. La optimización de escenarios multimodales o funciones de fitness multimodales involucra características como el número total de óptimos en el espacio de búsqueda, la cantidad y calidad de los óptimos que se desean localizar, y el grado de dificultad del problema debido a aislamiento u óptimos confusos. No se van a examinar todos los posibles tipos de funciones multimodales, únicamente, nos centraremos en resolver un tipo general de problemas de optimización de funciones multimodales que

incluye (de alguna forma) los problemas prácticos típicos a los que se les aplican los métodos de nichos.

El objetivo final en la optimización de una función multimodal es encontrar varias soluciones, tanto globales como locales. Se debe asumir un espacio de búsqueda S y una función objetivo *f* que asigna un número real a cada elemento de *S*, (*f : S→ R*, donde *R* es el conjunto de los números reales). Se debe asumir además sin pérdida de generalidad, que el objetivo es la maximización de *f*.

Sean $S \subset R$, *f: S→ R* y i $\in$S. Se dice que la función *f* tiene un máximo relativo o máximo local, en el punto i, si existe un entorno *N(i)* tal que *f(x) <=f(i)* para todo x $\in$ *A∩N(i).* En este caso se dice que el punto *i* es un punto de máximo relativo, o local, de la función *f*.

Sea c el número de máximos locales y z al número de máximos globales (en el caso de que se pretendan optimizar varias funciones. Diversos tipos de problemas de optimización de funciones multimodales pueden resultar de interés para personas que resuelven problemas reales. A continuación se presenta un conjunto de estos problemas:

- Encontrar cualquier b < c máximos.
- Encontrar todos los c máximos.
- Encontrar por lo menos los b <= c máximos mayores.
- Encontrar cualquier b < z máximo global.
- Encontrar los z máximos globales. Son importantes en la optimización de funciones multiobjetivo, es decir que se disponen de varias funciones a optimizar.

Debe destacarse que al resolver un problema equivalente del tipo 2, se encuentra una solución a cualquiera de los otros tipos. Por ello se va a centrar el esfuerzo en encontrar los c máximos.

Existen otros tipos de problemas que no se encuentra en la lista pero que no son tan interesantes como encontrar el menor máximo. Una manera de manejar criterios secundarios es adicionarlos en la función de fitness.

4.3 Métodos para la formación de nichos

En esta sección se van a describir algunos de los métodos utilizados para la formación de nichos.

Iteración

La iteración es una de las técnicas más simple que puede ser usada con cualquier método de optimización para la obtención múltiples soluciones y puede incorporarse a cualquier método diseñado para la búsqueda de una única solución. Consiste en el acto de repetir un proceso con el objetivo de alcanzar una meta deseada, objetivo o resultado. Al aplicar el algoritmo de iteración a cualquier problema de solución simple, se puede obtener la mejor solución. En el caso de que exista un número p de soluciones, se deberá iterar como mínimo p veces más. Este algoritmo no garantiza que se localice todas las soluciones e incluso en diferentes iteraciones encuentre las mismas soluciones. Para compensar estos problemas se deberá aumentar en número de iteraciones por un factor de redundancia R. En el caso que todos los máximos tiene la misma probabilidad de ser localizado R se podría calcula como.

$$R = \sum_{m=1}^{p} 1/m$$

Para Jolley,1961 si el número de soluciones buscada es mayor que 3 entonces el valor de R se podría calcular como $R \approx \gamma + \log p$ donde $\gamma \approx 0.5777$ (constante Euler)

Paralelo sub-población

Otra técnica que puede obtener múltiples soluciones con AG cosiste en dividir la población en múltiples sub-poblaciones y que cada sub-población se desarrolle en paralelo. Si son sub-poblaciones independiste se debería iterar de forma independientes las sub-poblaciones. No existe ninguna garantía de obtener las diferentes soluciones deseables. Para resolver este problema debería existir algún grado de comunicación entre las diferentes sub-poblaciones para permitir que los mejores individuos puedan extenderse a las diferentes poblaciones. Sin embargo esta solución puede conllevar a que se reduzca la diversidad de soluciones y la población completa tarde o temprano converja en una solución única (Grosso,1985)

Fitness Sharing (Método de proporción):

La idea de este método es la formación de subconjuntos de elementos vecinos en las poblaciones llamados nichos. De esta forma muchos óptimos pueden ser explorados al mismo tiempo. Teóricamente, el número de individuos residiendo cerca de un óptimo será proporcional a su valor. Se realiza degradando el fitness de cada individuo en la población por una determinada cantidad calculada en base al número de individuos similares que existen en la población.

La función de fitness es modificada siguiendo la siguiente la expresión:

$$f_i^* = \frac{f_i}{\sum_{j=1}^{N} Sh(d(i,j))}$$

Siendo

$$Sh(d(i,j)) = \begin{cases} 1 - \left(\frac{d(i,j)}{\sigma_{share}} \right)^{\alpha} & Si \quad d(i,j) < \sigma_{share} \\ 0 & \text{Re}\, sto \end{cases}$$

Donde

d(i,j) Distancias entre los individuos i y j.

σ_{share} Radio de nicho: determina la pertenencia o no al nicho.

α Regulador de la pendiente de la función de sharing. Valores utilizados (1) y (2)

La función f* decrece en razón al número de elementos pertenecientes a su nicho contenidos en la población en un momento dado, sin embargo, crece en razón al valor de su función de evaluación. Cuando el nicho tiene un único individuo la función no se modifica ya que Sh(d(i,i))=1.

Este método tiene algunas desventajas como el tamaño de la población debe ser elevado, el tiempo adicional requerido para calcular los fitness modificado y la posibilidad que cada sub-población no converja en el valor óptimo ubicado.

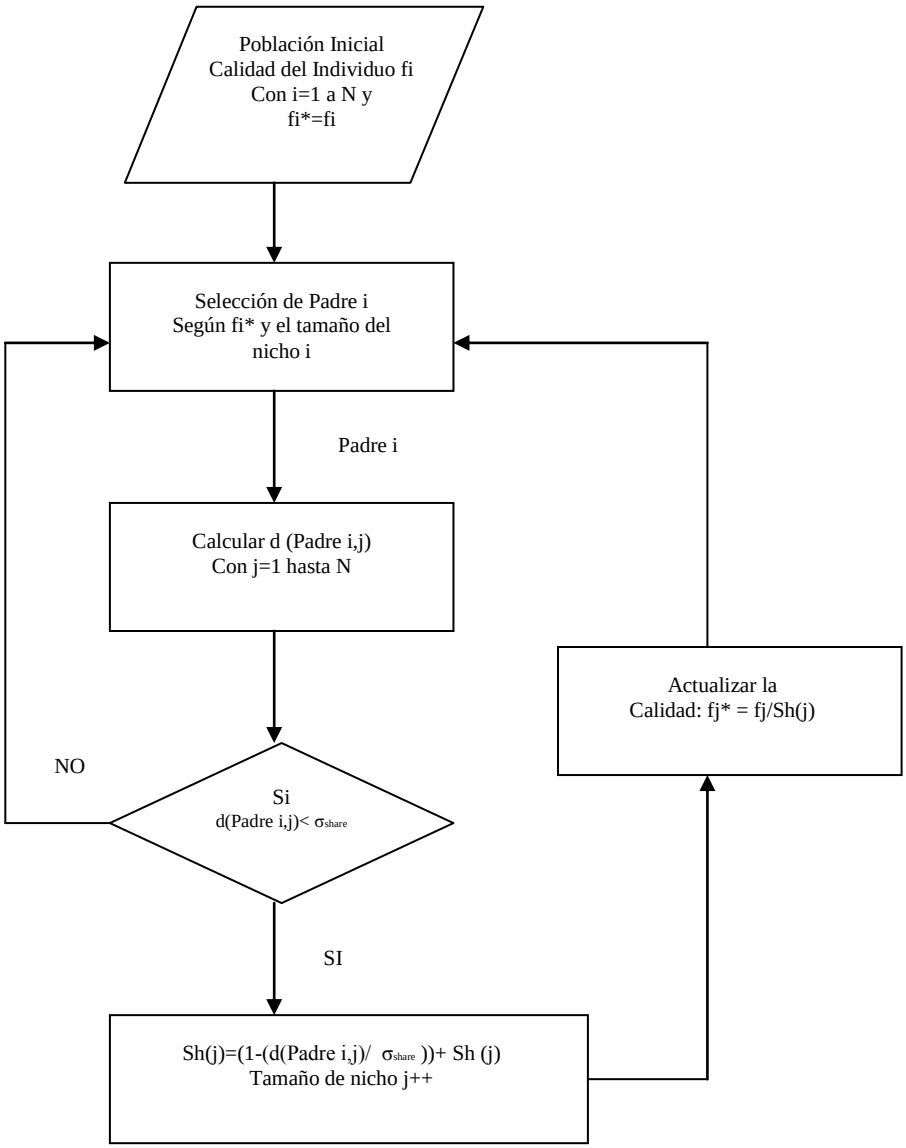
Constinuosly Updated Fitness Sharing (método de proporción con actualización continua):

Este proceso es una mejora del método original de Fitness Sharing. Propone actualizar el fitness de los individuos de la población en función de los padres que han sido seleccionados. Cada vez que se selecciona un padre, se actualiza el valor de adaptación de los elementos de la población.

En los estudios realizados se recomienda la selección de Torneo Binario: Cuando los dos individuos pertenecen a nichos poco poblados (menor que un tamaño de nicho máximo) el ganador es el que tiene la mejor adaptación. En caso contrario, el ganador es el individuo que tiene más riego de desaparecer (menor f*).

En la figura se muestran los pasos del algoritmo "Continuously Updated Fitness Sharing".

σ_{share} Radio de nicho: determina la pertenencia o no al nicho.



Clearing(Método de Aclarado)

El algoritmo realiza una selección únicamente sobre los individuos dominantes de la población. Se deben realizar los siguientes pasos:

1. Antes del proceso de selección se debe clasificar la población según la adaptación de forma decreciente.
2. Se debe elegir al primer individuo (el mejor) y de forma descendente se compara con el resto de la población. Aquellos individuos que están dentro de su radio de nicho (individuos dominados) se deben eliminar.
3. El proceso continúa con el segundo individuo clasificado, que aún no hay sido eliminado de la criba anterior. Y este eliminará los individuos dominados por él.
4. Finalizará el proceso cuando se obtenga los dominantes de cada nicho y con ello se realizará la selección.

El método de aclarado requiere de 2 parámetros. σ es el radio de nicho y "Kappa" es el número de individuos que se mantienen por nicho (los mejores). El algoritmo sigue la siguiente secuencia:

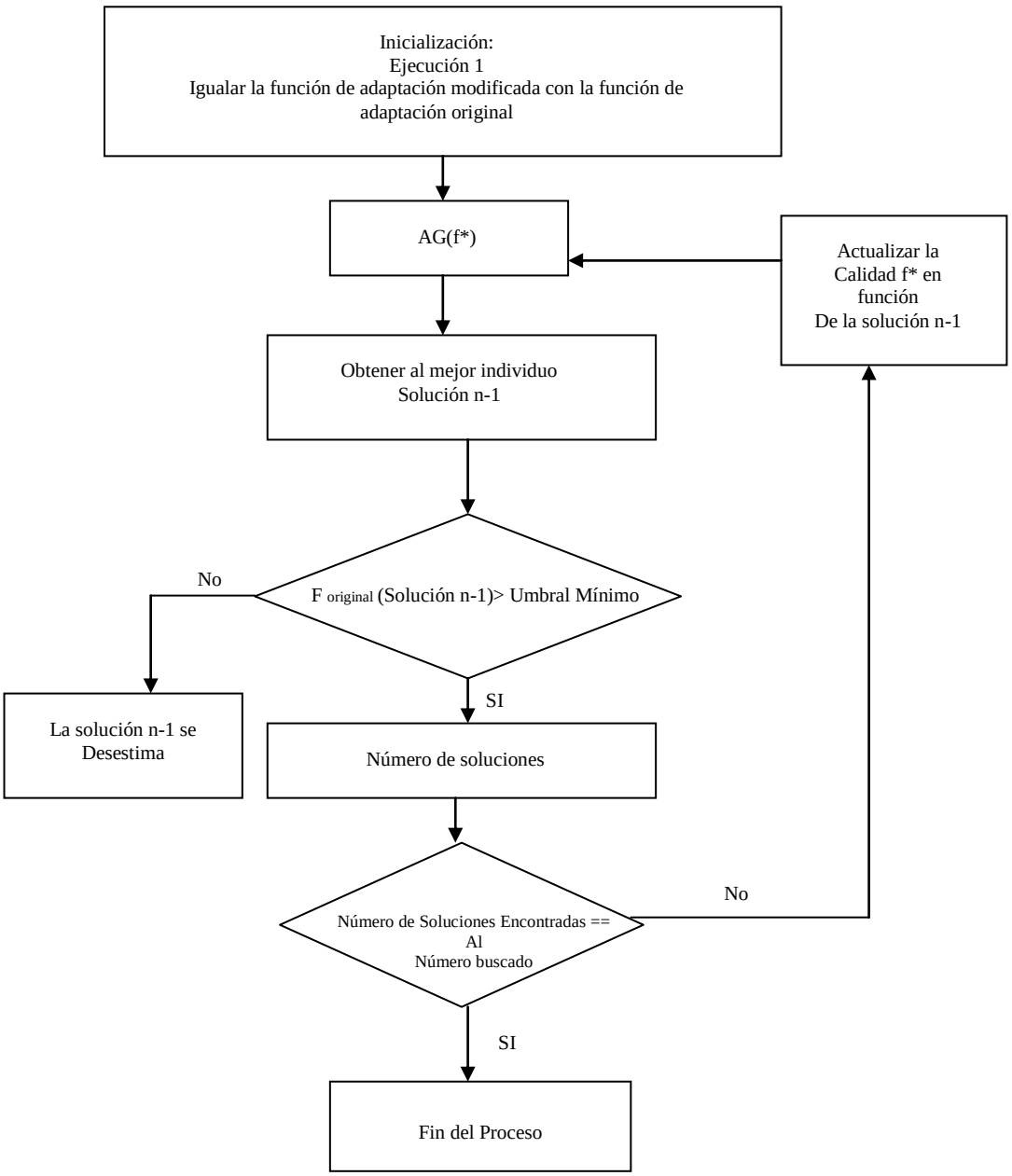
```
Ordenar P de mejor a peor
for i = 0 to N-1
{
        if ( Fitness(P[i] )>0)
        {
                NumGanadores=1
                for j =i+1 to N-1
                        if (Fitness (P[j])>0) and (Distancia(P[i],P[j])<σ))
                        {
                                if(NumGanadores<Kappa)
                                        NumGanadores++
                                Else
                                        Fitness(P[j]) = 0
                        }
        }
}
```

Sequential (Nichos Secuenciales):

Consiste en la ejecución secuencial de AG´s básicos de forma dependiente. En los primeros pasos del algoritmo se obtiene una solución, si su calidad está por encima de un umbral mínimo se guarda como solución del problema y se incremente el contador de soluciones halladas. Con esta solución, se modifica la adaptación de los individuos en los siguientes AG, penalizando de esta forma las zonas ya exploradas en el AG anterior. Se vuele a realizar una nueva ejecución, obteniendo otra solución. Para las siguientes ejecuciones se modificará la adaptación teniendo en cuenta todas las soluciones encontradas en pasos previos. El proceso termina cuando se tenga todas las soluciones deseadas.

Para los algoritmos secuenciales de nicho es necesario definir una distancia métrica, de modo que dados dos cromosomas, devuelva un valor relativo para ser comparado con otros. La distancia más trivial es la distancia de Hamming, aunque como Deb y Goldberg (1989) mostraron, la mejor opción sería la utilización de una medida de distancia basada en el espacio de soluciones descifrado. Esto es así porque puede ocurrir que la distancia entre dos cromosomas binarios no se corresponda con la distancia real en el espacio de soluciones. Algunas desventajas de este algoritmo son que puede llegar a ser complicada su programación y puede llegar a requerir bastante tiempo de ejecucin.

Un esquema del algoritmo se muestra en la siguiente figura:



Se va modificando la función de fitness, f*(x) para un individuo x, esta función está influenciada por el fitness inicial de la función. Inicialmente la $f_o^*(x) = f(x)$ .Y en cada ejecución, las mejores soluciones encontradas S_{n-1} crean una función de liberación $G(x, s_{n-1})$ que modifica la función de fitness para las siguientes ejecuciones:

$$f_n^*(x) = f_{n-1}^*(x) * G(x, s_{n-1})$$

Existen dos formas para calcular la función c: fórmula potencial G_p y exponencial, las cuales vienen dadas por:

$$G_p(x, s_{n-1}) = \begin{cases} \left(\dfrac{d(x, s_{n-1})}{\sigma_{share}} \right)^{\alpha} & Si \quad d(x, s_{n-1}) < \sigma_{share} \\ 1 & \text{Re}\, sto \end{cases}$$

$$G_e(x, s_{n-1}) \begin{cases} \exp(\log m * \dfrac{\sigma_{share} - d(x, s_{n-1})}{\sigma_{share}} R) & Si \quad d(x, s_{n-1}) < \sigma_{share} \\ 1 & \text{Re}\, sto \end{cases}$$

Siendo σ_{share} el radio del nicho, $d(x, s_{n-1})$es la distancia entre x y s_{n-1} determinado por una distancia métrica. x es un individuo de la ejecución n, y s_{n-1} es la solución encontrada en la ejecución n-1. α es el factor de poder, el cual determina

La función sharing descrita por Goldberg y Richardson (1987), Deb. (1989) y Deb y Goldberg (1989) trata de reducir el fitness de los individuos dependiendo de la distancia a otros individuos en la población. De un modo similar, la función descrita anteriormente, trata de reducir el fitness de un individuo dependiendo de su distancia de cada individuo mejor, encontrado en ejecuciones anteriores.

Otros métodos

Glover (1989) describe un método conocido como "tabu search" búsqueda tabú es un método de optimización, pertenecientes a las clase de técnicas de búsqueda local. La búsqueda tabú aumente el rendimiento del método de búsqueda local mediante el uso de estructuras de memoria: una vez que una potencial solución es determina se la marca como "tabú" de modo que el algoritmo no vuelve a visitar esa posible solución. La búsqueda tabú utiliza un procedimiento de búsqueda local para moverse iterativamente desde una solución x hacía un solución x', hasta satisfacer algún criterio de parada. La búsqueda tabú modifica la estructura de vecinos para cada solución a medida que la búsqueda progresa.

Smith et al. (1992) describen una técnica que puede promover la formación de nichos en un sistema clasificador. Demuestran que es similar al fitness sharing, pero evita las limitaciones que deben conocer el número de máximos, que los máximo deben ser extendidos y que hay un componente de complejidad de tiempo $O(N^2)$. Lamentablemente su técnica es sólo aplicable a clasificación de sistemas, y no para optimización de función multimodal

4.4 Problemas en los algoritmos con nicho

Los algoritmos con nichos dan buenos resultados con funciones trampas, además localiza los máximos locales, requieren menos ejecuciones que las iteraciones ciegas. Pero es necesario tener en cuenta a la hora de crear estos algoritmos los siguientes problemas:

Problema con radio nicho:

La inapropiada elección del radio puede conllevar soluciones erróneas. Si se utiliza un radio de nicho pequeño se puede dividir la población en múltiples nichos pudiendo ser contraproducente para el funcionamiento correcto de los AG. Puede llevar a mecanismo de selección elitista, al buscar distinto nicho la misma solución, perdiendo la diversidad de la población, y converge de forma prematuramente a una única solución. Si por lo contrario se utiliza un radio de nicho grande puede encontrarse diferentes soluciones en un mismo radio del nicho.

Para solucionar estos problemas seria encontrar el valor del nicho correcto. O una utilización conjunta de los radio del nicho. Se debería la utilización de radio de nichos grandes para aproximarse a las soluciones y para localizar con exactitud y encontrar los máximos perdidos debería utilizarse nicho de radio más pequeño.

Otros problemas pueden ser identificados en la puesta en práctica de los algoritmos. Éstos dan lugar a tres síntomas:

- Inexactitud: Las soluciones no son completamente exactas
- Incompetencia: No encuentra todas las soluciones en las distintas secuencias.
- Extra de Generaciones: A menudo requieren más ejecuciones del algoritmo para encontrar las posibles soluciones.

Inexactitud

Esto es un problema estándar con todo los AG (DeJong 1985). Una posible solución es usar un método híbrido en el cual un AG encuentra soluciones aproximadas, luego usar una técnica local de búsqueda, para dar un resultado más exacto (Goldberg, 1989; Davis 1991). Este problema es parecido a cuando se usa un radio de nicho demasiado grande

Incompetencia

Esto es principalmente un efecto secundario de localizar soluciones muy inexactas. Una de las principales dificultas de los AG es identificar correctamente las mejores soluciones. Existen en algunas ejecuciones que a menudo el máximo no será contado como una solución, ya que el fitness en este punto está por debajo del umbral de solución. Y otras que intenta a converger en soluciones de máximos que son inexistentes. Estos errores además producen que cuando se modifique la función fitness de por la función de liberación sea incorrecta. Como consecuencia de este error puede que los máximos verdadero tenga menos probable de ser encontrada en posteriores ejecuciones, al modificar la función de fitness con soluciones incorrectas.

Según el tipo de problemas que se busque solucionar a veces es mejor tener un algoritmo con una probabilidad de fallo del 25 %, que tener siempre una cuota alta de éxito pero mucho más lento en su ejecución . Y la segunda solución sería propuesta en la de inexactitud, usar una técnica local de búsqueda. Este acercamiento soluciona ambos problemas inmediatamente, sin necesariamente añadir enormemente al número de evaluaciones requeridas

Extras Generaciones.

Hay varias causas en las que los algoritmos deben seguir ejecutándose creando más generaciones para encontrar las soluciones.

En primer lugar, puede ser causado por la convergencia hacia un máximo local no deseado. Las ejecuciones no siempre localizan un máximo con un fitness por encima del umbral de solución, y en ese caso el máximo encontrado no se cuenta como una solución. La convergencia sobre el máximo local, es un problema estándar. Una solución sería la utilización de algoritmos nicho secuencial al no convergen en los máximos ya encontrados en anteriores ejecuciones. Aunque no es la manera más eficaz de encontrar todos los máximos interés, pero nos aseguramos de encontrar los máximos.

Y la última causa que se tenga que ejecutar generaciones extras es que converger en un máximo local incorrectos ya sea por causas al modificar la función de fitness o la utilización de radio pequeños para encontrar máximos a igual altura pero desigual anchura.

Capítulo 5. Librería Desarrollada

En este apartado, se explicará la librería escrita para utilización de algoritmos genéticos con nichos en la resolución de problemas multimodales. Para generarla se buscó un lenguaje de programación que soportase programación orientada a objetos, fuese estructurado, utilizase un framework fácil para realizar depuraciones, que posea una librería con una amplia documentación actualizada y disponible para la programación. Por eso se decidió realizar la librería en C++.

La librería generada se ha basado en algunos conceptos del Fitness Sharing (método de proporción) como se indica a continuación

- La librería modifica a los individuos para que pertenezcan a la subpoblación que crea los nichos. Estas modificaciones se realizarán en las últimas generaciones. Y los valores de los individuos serán un valor aleatorio que pertenezca al rango que acota los puntos del nicho.
- También aquellos nichos que tenga una población más elevada que el resto no permitirá incluir más individuos a su subpoblación favoreciendo que el individuo debe interactuar con el resto de los nichos. La libraría utilizará un contador de individuos de cada subpoblación. Y hasta que el resto de nichos no tenga el mismo número de individuos, será una zona prohibida no pudiendo pertenecer a esa subpoblación hasta que se igualen el resto.
- Se utiliza la distancia de dos individuos y una constante llamada radio de nicho para indicar la pertenencia o no a un nicho.
- Un nicho estará asociado con la búsqueda de un óptimo.

La librería que se ha implementado está diseñada para encontrar y conservar todos los posibles puntos o zona del espacio de búsqueda que correspondan a óptimos locales u óptimos globales. Para obtener este resultado se debe permitir la búsqueda simultánea en diferentes áreas del espacio de búsqueda. La librería intentara mantener diferentes subpoblaciones en todas las posibles soluciones. Y cada subpoblación estará acotada por los dos puntos que forman un nicho. Deberán existir tantos nichos como soluciones se busquen a un problema. Y cada nicho debe buscar una solución diferente.
La creación de la librería se apoya en la utilización de una lista enlazada como se muestra en la figura 5.1. Una de las ramas conectará los nichos y la otra rama unirá los dos puntos que forman un nicho. Los puntos son los extremos que forman el nicho. La subpoblación del nicho está acotada por esos valores, siendo los puntos el límite inferior y superior. Existirán problemas de alta complejidad que con dos puntos no se podrá acotar la población. La librería utiliza estructura dinámica en la formación del nicho para estos casos, pudiéndose utilizar más de dos puntos para ir acotando la subpoblación. Como mínimo un nicho tiene que está formado por dos puntos. Lo ejemplos que se ha utilizado para probar la librería han sido búsqueda de máximo locales y globales de unas funciones y se han empleado solamente dos puntos en cada nicho para acotar la subpoblación. Para futuros trabajos se podría implementar problemas de alta complejidad que necesitasen más de dos puntos para acotar a la subpoblación y ver cómo responde la librería.

Un punto está constituido por los siguientes campos:

- Value: Tabla de enteros, donde se almacena el valor del cromosoma
- Fitness-Real: Variable para almacenar el valor que tiene el cromosoma para la función utilizada.

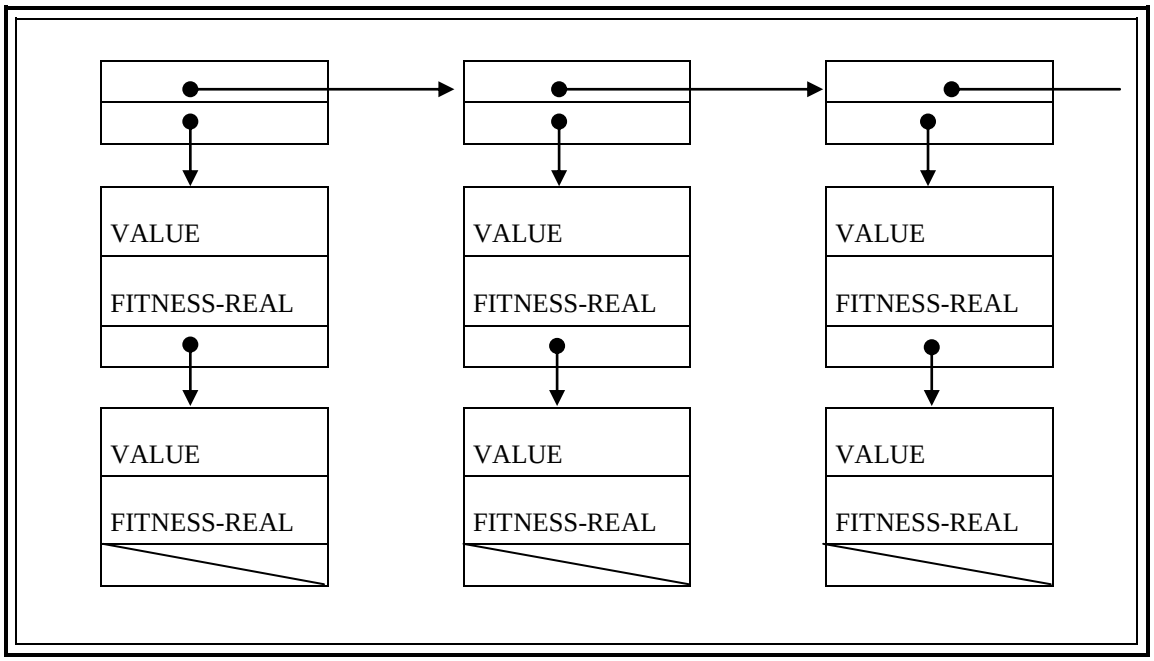


Figura 5.1 Esquema de la estructura utilizada

A continuación se muestra la estructura definida por la librería para la lista enlazada. Está formada por dos punteros y se representará de esta forma:

```
struct lista_nichos {
        struct nichos *elemento;
        struct lista_nichos *sig;
};
```

Y para cada punto del nicho, la estructura es la siguiente:

```
struct nichos {
        long int *value;
        double fitness_real;
        struct nichos *sig;
};
```

Además de los objetos creados, se utilizan las siguientes funciones:

- Constructor de la clase: Se reserva memoria para un determinado número de nichos introducidos en un fichero de configuración. Para poder definir los parámetros iniciales se ha definido un fichero de configuración que se deberá adaptar a las necesidades de cada problema.
- Destructor de la clase: Borra todos los nichos creados, liberando la memoria utilizada.

- Modificación de nichos: Según los individuos que se obtienen en cada generación, irán modificando los puntos de los nichos, para ir acotando las soluciones al problema.
- Limpieza de nichos: La siguiente función modifica y elimina los nichos que no sirven para llegar a la solución óptima. Borrando los valores y liberando la memoria utilizada.
- Inserta nichos: Cuando los puntos del nicho poseen un fitness elevado y se encuentran distanciados, se crea un nuevo nicho que separará los dos puntos.

La librería se iniciará creando un número determinado de nichos y los valores almacenados en los puntos del nicho serán el límite inferior y el superior del espacio de búsqueda del problema.

Una vez inicializados los nichos, el siguiente paso es generar la población obteniendo los individuos con su correspondiente valor de fitness. Esta primera generación será obtenida de forma aleatoria o a través de la lectura de un fichero. A continuación los individuos interactuarán con los nichos. Se modificarán los valores del nicho para ir reduciendo el espacio de búsqueda. El valor de un individuo sólo podrá modificar el valor del nicho más cercano. La librería buscará el nicho que esté más cerca del individuo y se encuentre entre sus valores. Si el individuo no se encuentra entre los valores de ningún nicho se busca el más cercano. Una vez localizado el nicho se compara los valores del fitness del nicho con el del individuo. Si algún valor del nicho es inferior al de individuo será sustituido el menor de los valores del nicho por el individuo.

Cuando ya se ha tratado gran parte de la población, se obtendrá una nueva población gracias a los métodos de AG(selección, cruzamiento, mutación). Con los nuevos individuos se vuelve a interactuar con la estructura de nichos. Este paso se repetirá hasta que se cumpla una condición de salida que dependerá del problema que se esté tratando. De esta forma se consigue obtener las mejores zonas de subespacio comparándolas con sus vecinos. Y con el paso de generaciones, esas zonas deben ir mejorándose al acercarse más a una de las soluciones buscadas.

En las primeras generaciones se intenta que los nichos creados acoten todas las posibles soluciones al problema y no se modificará ningún individuo para obtener más diversidad en las siguientes generaciones. Una vez sobrepasado un número determinado de generaciones y estén definidos los nichos, se modifican los valores de los individuos para que pertenezcan solamente a la subpoblación que acota por los nichos. Además si en las últimas generaciones una subpoblación está muy densamente poblada se intentará distribuir los individuos a otras subpoblaciones que estén menos pobladas.

También a partir de un número elevado de generaciones se eliminarán aquellos nichos que no estén buscando ninguna solución correcta, eliminándose los nichos para los que las distancias de sus puntos estén alejados y posean un fitness inferior a la media de la población, y también se eliminarán aquellos nichos que estén examinando una misma solución quedándose la librería sólo con uno. E incluso puede que se cree un nuevo nicho si los puntos que forman un nicho están alejados y poseen un fitness elevado

PSEUDOCÓDIGO:

```
Inicio
Leer fichero Inicializar variables
Num_generacion←0
Hacer Mientras  MetodoParada && num_generacion<100
  Num_generacion← Num_generacion+1
  Poblacion.iterar        // se realiza toda las operaciones con los individuos que forma la
                          // población  Selección, Cruzamientos y  mutación
  Poblacion.ordenar    // se ordena la población de mayor a menor
  Repetir  i←1 hasta i<=Población.tamaño_poblacion
    Nicho.Insertar_nichos(Poblacion[i].Valor_individuo, Poblacion[i].Valor_fitness)
    Si num_generación> Condición_Cambio
          Poblacion.cambiar_cromosomas(Poblacion[i].Valor_individuo)
    Fin si
    Si  num_generacion>num_generacion_modificar // A partir de una generación se
                                                  //eliminara algún nicho
                                                  // y se modificaran valores
          nich.ajustar_nichos();
          nich.eliminar_nichos_hueco(media_fit);
          nich.crear_nicho(media_fit);
    Fin si
  Fin Repetir
Fin Hacer

Fin
```

5.1 Funciones

Seguidamente se explican las principales funciones utilizadas para desarrollar la librería.

FUNCIÓN: int* **Insertar_nichos** (long int valor_cromosomas, double &fitness)

ENTRADA: valor_cromosomas: es el valor de un individuo que se va comparar con los nichos y fitness: es el valor de fitness de ese individuo.

SALIDA: La función devuelve un cero si se ha modificado algún valor del nicho y un 1 si no se modificado

ALGORITMO: El primer paso es buscar el nicho que esté más cerca del individuo y que se encuentre entre sus valores. Si el individuo no se encuentra entre los valores de ningún nicho, se busca el más cercano. Una vez localizado el nicho se compara el fitness de los puntos almacenados con los del individuo. Si uno de los puntos del nicho posee un fitness menor entonces será sustituido por el individúo de entrada y su correspondiente fitness devolviendo un "0" y si no se ha modificado ningún nicho devolverá un "1"

PSEUDOCÓDIGO:

```
INICIO
Puntero←Inicio nichos
Min_distancias←MAXLONG //introduce el número máximo posible
Lejos←1
Repetir  i←1 hasta i<=NumNichos  && salida
    entreNichos <-entre_valores(valoreCromosomas,puntero);//0 fuera del nicho 1 dentro
    Si entreNichos ==1 //entre los valores del nicho
         Auxilar_dist←distancia_valores(valor_cromosomas,puntero)
        Si(Auxiliara_dist<Min_distancia)
            Min_dis←Auxiliar_dist
            NumNicho←i
        Fin si
    Fin si
    Else//entre==0 No está entre los valores del nicho
            Auxiliar_dist=distancia_valores(valores_cromosomos,putero);
            Si (Min_distancias > Auxiliar_dist)
                  Min_distancias ←auxiliar_dist
                  NumNicho←i
            Fin si
      Fin else
Fin Repetir
//Nos posicionamos en el nicho seleccionado
For (i=1, i< NumNicho,i++)
         Puntero←Puntero.sig
Fin For
valor←Cambiar_nichos(puntero,valor_cromosomas,fitness)
return valor
FIN
```

FUNCIÓN: int **entre_valores**(long int * valorA,struct lista_nichos *nicho);
ENTRADA: valorA: el valor del cromosoma y lista_nichos : nicho que se va a comparar
SALIDA: la función devuelve un valor entero, si es "1" indicará que el valor del cromosoma está entre los valores del nicho, si es "0" esta fuera del rango y "-1" existe un error.

ALGORITMO: Se obtienen las coordenadas cartesianas del individuo y de los puntos que forman el nicho. Se comprueba si el valor del individuo se encuentra dentro del rango que forman los dos puntos del nicho devolviendo un 1 si está dentro y 0 si está fuera.

PSEUDOCÓDIGO:

```
Inicio
    // Obtener valores cromosoma
    SacaValores(xValores ,yValores, valorA)
    // Obtener valores del primer objeto del nicho
    SacaValores( x Primero,y Primero, nicho.value)
    // Obtener valores del segundo objeto del nicho
    SacaValores(xSegundo,ySegundo,nicho.sig.value)

    Si (xValores >= xPrimero  and xValores <= xSegundo
        and  yValores>= yPrimero and yValores<= ySegundo)
          //Entonces está entre los valores nicho
                Return 1;
          Sino Si (xValores >= xSegundo  and xValores <= xPrimero
              and  yValores>=  ySegundo and yValores<= yPrimero)
                //Entonces está entre los valores nicho
                Return 1;
          Sino
                //Valor no está entre nicho
                Return 0;
          Fin Si
    Fin Si

Fin
```

FUNCIÓN: double **distancia**(long int *num_nichos,long int *numero);
ENTRADA: los dos valores de los cuales se quiere obtener la distancia
SALIDA: Da como resultado la distancia entre los dos valores.
ALGORITMO: Calcula la distancia euclidiana entre los dos valores. Primero obtiene las coordenadas cartesianas de los dos valores entrada, y a continuación se calcula la distancia entre ellos, con la siguiente fórmula:

$$\sqrt[2]{((U_1 - V_1)^2 + (U_2 - V_2)^2)} = distancia$$

PSEUDOCÓDIGO:

```
Inicio
    SacaValores(xValorPrimero ,yValorPrimero, num_nichos)
    SacaValores( xValorSegundo,yValorSegundo, numero,)
    valorx← xValorSegundo- xValorPrimero
    valory← yValorSegundo-yValorPrimero
    total←(valorx*valorx)+(valory*valory)
    total←sqrt(total) // Raiz cuadrada,
Fin.
```

FUNCIÓN int **cambiar_nichos**(lista_nichos *puntero,long int *val_cromosomas, double fitness)
ENTRADA: lista_nicho es el nicho seleccionado, val_cromosoma es el valor del cromosoma y el fitness el valor de fitness del cromosoma.

SALIDA: La función devuelve un valor entero. Si el valor que se devuelve es un 0 se ha modificado el nicho. Si es 1 indicará que el nicho no ha sido modificado, porque los valores almacenados en el poseen un fitness superior al individuo, y un -1 si se ha producido un error

ALGORITMO: El fitness de entrada se compara con los valores almacenados en el nicho. Si el valor es menor que el del nicho se devuelve un 1, si no se modificará el menor valor por el individuo de entrada y se devolverá un cero

PSEUDOCÓDIGO:

```
INICIO
   Si (fitness >lista_nicho.fitness o fitness>lista_nicho.sig.fitness)
        Si (lista_nicho.fitness >lista_nicho.sig.fitness)
                lista_nicho.sig.fitness←fitness
                lista_nicho.sig.value←value=val_cromosomas
        sino
                lista_nicho. fitness←fitness
                lista_nicho..value←value_cromosomas
        Fin si
   Return 0
   Fin si
Return 1
FIN
```

FUNCIÓN long int* **cambiar_cromosomas**(Valor_individuo)
ENTRADA: valor_individuo Este individuo no pertenece a ningún nicho y cambiará su valor por algún valor aleatorio del nicho que tenga menos población.
SALIDA: Será un valor aleatorio que esté dentro del rango de un nicho
ALGORITMO: Se busca el nicho que tenga menos población. Una vez localizado se se devolverá un valor aleatorio de dicho rango.
PSEUDOCÓDIGO:

```
Inicio
   listaNicho ←inicio
   NumNichos←listaNicho.tamaño()
   Aux←1;
   Num_poblacion←listaNicho.numPlobación//No indica num individuo tiene el nicho
   //Se busca el nicho con menos población
   Repetir      i←1 hasta i<=NumNichos
        Si Num_poblacion> listaNicho.numPlobación
           Aux←i
           Num_poblacion ←listaNicho.numPlobación
        Fin Si
```

```
        listaNicho←listaNicho->sig
    Fin Repetir
    listaNicho ←inicio
    // No situamos en el nicho con menos población
    Repetir      i←1 hasta i<=Aux
      listaNicho←listaNicho->sig
    Fin Repetir
// Sumamos uno más a la población de ese nicho y devolvemos un valor aleatorio
listaNicho.numPlobación← listaNicho.numPlobación+1
return  saca_aletorio(listaNicho.elemento, listaNicho.sig. elemento)
Fin
```

FUNCIÓN: void **ajustar_nichos**(void)
ENTRADA: Nada
SALIDA: Nada
ALGORITMO: Si varios nichos están buscando la misma solución, se quedará solo con un nicho y los mejores puntos para encontrar la solución. Para conseguir este propósito la función va recorriendo todos los nichos y se comparará la distancia con el resto. Aquellos nichos que estén muy cerca, seguramente están buscando la misma solución, entonces se quedará sólo con un nicho y eliminará el resto de nichos que estén buscando la misma solución.
PSEUDOCÓDIGO:

```
Inicio
Puntero←inicio
Repetir          i hasta i<=NumNichos
    Auxiliar←inicio
    ValorrA←Puntero.valorAfitness
    ValorB←Puntero.valorBfitness
    Repetir j hasta j<=NumNichos
        Si (Auxiliar<>Puntero)
            lejosA←lejos_nicho(valorA,auxiliar)
            lejosB←lejos_nichos(valorB,auxiliar)
            Si      !lejosA && !lejosB      && valorA<=auxiliar.Valorfitness      &&
            valorB<=auxiliar.ValorFitness
                    Borrar_Nichos (auxiliar)
            Fin si
            Else
                Si  !lejosA
                    Si numA>auxiliar.elemento.sig.fitness_real
                        Auxliar.elemento←numA
                    Fin si
                    Else
                        Si numA>auxiliar.sig.fitness_real
                            Auxiliar.sig←numA
                        Fin si
                    Fin else
                Fin si
                Si  !lejosB
                    Si numB>auxiliar.elemento.sig.fitness_real
```

```
                    Auxliar.elemento←numB
                Fin si
                Else
                    Si numB>auxiliar.sig.fitness_real
                        Auxiliar.sig←numB
                    Fin si
            Fin
        Fin Si
       Fin Else
    Fin si
Auxiliar←Auxiliar.sig
Fin Repetir
Puntero←Puntero.sig
Fin Repetir
```

FUNCIÓN: void **eliminar_nichos_hueco**(double avg_fitness);
ENTRADA: Introduce el valor medio de los fitness de la población.
SALIDA: No se devuelve nada, pero modificará la lista de nichos.
ALGORITMO: Los nichos cuyos fitness sean inferiores al valor introducido y su distancia sea superior a la a la constante distancia, serán eliminados de la lista de nichos liberando memoria.
PSEUDOCÓDIGO:

```
Inicio
   NumNichos←listaNicho.tamaño()
   Repetir      i←1 hasta i<=NumNichos
        dist←distancia(listaNichos[i].valorPrimero listaNichos[i].valorSegundo)
        Si( num_medio_fitness >listaNichos[i].valorPrimero y
          num_medio_fitness >listaNichos[i].valorSegundo y
                dist> constantes_distancias)
                Borrar nichoSelecionado
                Num_nichos--;
        Fin si
   i++
   Fin Repetir
Fin
```

FUNCIÓN: int **crear_nicho**(double avg_fitness)
ENTRADA: Introduce la media de los fitness reales de la población.
SALIDA: Saca un valor entero indicando si se ha realizado correctamente o si se ha producido algún error al crear unos nichos -1.
ALGORITMO: Primero se recorren todos nichos y se calcula la distancia entre los dos puntos del nicho. Si la distancia es superior a la constante y poseen ambos puntos un fitness superior a la media de la población, se separan los dos puntos en dos nichos. Y cada nicho tendrá el valor de un punto y en el otro punto un valor aleatorio
PSEUDOCÓDIGO:

```
Inicio
   Num_medio_fitness← Num_medio_fitness+1
   NumNichos←listaNicho.tamaño()
   Repetir      i←1 hasta i<=NumNichos
```

```
        Se calcula la distancia entre os puntos del nicho
         Dist←Distancias(listaNicho[i].puntoA,listaNicho[i].puntoB)
   Si ( dist> constantes_distancias y listaNicho[i].fitnessA>num_medio_fitness y
   listaNicho[i].fitnessA>num_medio_fitness)
   Entoces
        //Dividir el nicho seleccionado en dos Nichos
            NichoA.valorPrimero←Nicho_selecionado.valorPrimero
           aux←Nicho_selecionado.valorPrimero;
           NichoA.valorSegundo← saca_aleatorio_Cerca(aux);
           NichoB.valorSegundo←Nicho_selecionado.valorSegundo
           aux2← Nicho_selecionado.valorSegundo;
           NichoB.valorPrimero← saca_aleatorio_Cerca(aux2);
        Sino i++;
        Fin Si
   Fin Repetir
Fin
```

FUNCIÓN: void **ver_nichos**(void)
ENTRADA: nada
SALIDA: nada
ALGORITMO: Muestra por pantalla los valores de los nichos. Recorriendo el nicho e imprimiendo los valores
PSEUDOCÓDIGO:

```
Inicio
   NumNichos←listaNicho.tamaño()
   Repetir      i←1 hasta i<=NumNichos
        Impirmir(listaNIcho)
        listaNicho←listaNicho->sig
   Fin Repetir
Fin
```

FUNCIÓN: int **lejos_nichos**(long int * valorA,struct lista_nichos *nicho);
ENTRADA: Introduce el cromosoma y el nicho, que se compararán.
SALIDA: La función devuelve un cero si están cerca y uno si está lejos.
ALGORITMO: Se calcula la distancia entre el individuo de entrada con los puntos de los nichos. Si una de esta distancia es inferior o igual a una constante (radio del nicho) devuelve un cero, si no, un uno.
PSEUDOCÓDIGO:

```
Inicio
   distanciaPuntoA ←CalculaDistancia(Cromosomas,nicho.primerValor)
   distanciaPuntoB← CalculaDistancia(Cromosomas,nicho.segundorValor)
   Si (distanciaPuntoA<=constantes_distancias  o
        distanciaPuntoB<=constantes_distancias)
        Entoces  return 0
   Fin Si.
   Sino
        Return 1;
   Fin Sino
Fin
```

FUNCIÓN: long int * **saca_aletorio**(long int *valorA,long int *ValorB);
ENTRADA: Introduce dos valores para calcular un valor aleatorio
SALIDA: Devuelve como resultado un valor aleatorio entre los valores introducidos
ALGORITMO: Primero se descomponen en coordenadas cartesianas los valores introducidos. A continuación se llama a la función sacar_ValoresAleatorio para obtener las coordenadas aleatorias. Una vez obtenidos los valores, se transforman esas coordenadas en la cadena binaria
PSEUDOCÓDIGO:

```
Inicio
   SacaValores(num_nichos,xValorPrimero ,yValorPrimero)
   SacaValores (numero,xValorSegundo,ValorSegundo)
   coordenasAleatorioX← saca_ValorAleatorio(xValorPrimero,xValorSegundo)
   coordenasAleatorioY← saca_ValorAleatorio(yValorPrimero,yValorSegundo)
   return encode_valores (coordenasAleatorioX,coordenadasAleatoriaY)
Fin
```

FUNCIÓN: int **saca_aleatorio_Cerca**(double valorA);
ENTRADA: Introduce un valor
SALIDA: Devuelve como resultado un entero cercano al valor introducido
ALGORITMO: Se calcula un número aleatorio cercano al valor usando la costantes-distancia
PSEUDOCÓDIGO:

```
Inicio
   Si (rand()%0)
      Return saca_ValorAletorio(valoA,valorA+costantes_distancias)
   Else
     Return saca_ValorAletorio(valoA,valorA-costantes_distancias)

Fin
```

FUNCIÓN: **saca_ValorAletorio** (double ValorA , double ValorB)
ENTRADA: valorA ,ValoB se calcula un número aleatorio entre los dos valores
SALIDA: Devuelve como resultado un entero aleatorio
ALGORITMO: Se calcula un número aleatorio entre los dos valores de entrada.

PSEUDOCÓDIGO:

```
Inicio
   Si valorA>valorB
       Return  (valorB + rand() % valorA)
   Else
      Return (ValorA+rand()%ValorB)
Fin
```

FUNCIÓN: **sacar_valores** (double x,double y, long int cromosoma)
ENTRADA: long int cromosoma: introduce una cadena binaria para descomponerla en componentes cartesianas
SALIDA: Devuelve las coordenadas cartesianas de x e y
ALGORITMO: Se divide la cadena binaria en dos subcadenas. La primera subcadena será para obtener la coordenada X y la segunda para obtener la coordenada Y.
PSEUDOCÓDIGO:

```
Inicio
    CadenaX←cromosoma[0, cromosoma.length()/2];
    CadenaY←cromosoma[cromosoma.length()/2, cromosoma.length()];
    x←sacar_valores(CadenaX);
    y←sacar_valores(CadenaY);
Fin
```

FUNCIÓN: double **sacar_valores** (long int cromosoma)
ENTRADA: long int cromosoma: introduce el valor de un cromosoma para descomponerla en un valor entero
SALIDA: Devuelve el número entero de la cadena introducida
ALGORITMO: Se transforma la cadena binaria en un número entero
PSEUDOCÓDIGO:

```
Inicio
    valor ←0
    For (i←0;i<cromosoma.length()/2;i←i+1)
        valor ←cromosoma[i]*pow(2,i)+ valor;
    Fin For
    Return valor;
Fin
```

FUNCIÓN: char* **encode_valores** (double coordenasAleatorioX, double coordenadasAleatoriaY)
ENTRADA: coordenadas X e Y del individuo
SALIDA: devuelve la cadena codificada en binaria
ALGORITMO: Transforma las coordenadas cartesianas del individuo a una cadena binaria.
PSEUDOCÓDIGO:

```
Inicio
    Cadena1←encode_valor(X, longitud_cromosoma/2);
    Cadena2←encode_valor(Y, longitud_cromosoma/2);
    Return Cadena1+Cadena2
Fin
```

FUNCIÓN: char* **encode_valor** (double valorEntrada, double longitud_cromosoma)
ENTRADA: valorEntrada número entero que se quiere pasar a binario, longitud_cromosoma la longitud que tiene que tener la cadena devuelta.
SALIDA: devuelve la cadena codificada en binaria
ALGORITMO: El procedimiento consiste en ir dividiendo el valor entero por una base. Y así sucesivamente hasta que el cociente no sea divisible por la base. La cadena será la concatenación del último cociente y todos los restos desde el último hasta el primero.
PSEUDOCÓDIGO:

```
Inicio
    i←1;base←2;
    valor ← valorEntrada
    Hacer Mientras( (val← valor /base)>=base)
        Cadena1[(longitud_cromosoma ) - i] ←valor % base;
        i←i+1
        valor ←val
    Fin Hacer
    Cadena1[(longitud_cromosoma ) - i] ← valor % base;
    i←i+1

    Si (valor / base != 0)
         Cadena1 [(longitud_maxima) - i] = valor / base;
         i←i+1
    fin Si
    Hacer Mienstra (i<=longitud_maxima)
        Cadena1 [(longitud_maxima) - i] = 0;
        i←i+1
     Fin Hacer
    Return Cadena1
Fin
```

Capítulo 6. Pruebas Experimentales

6.1 Explicación librería con ejemplos.

En este apartado se explica con ejemplos como actúa la librería en los diferentes casos de uso.

A El individuo está entre los valores de un nicho			
A001	El individuo posee el fitness mayor que los valores del nicho	A011	El nicho acota a un máximo
		A021	El nicho acota a más de un máximo
		A031	El nicho no acota ningún máximo
A002	El individuo posee el fitness mayor de sólo uno de los valores almacenados del nicho.	A012	El nicho acota a un máximo
		A022	El nicho acota a más de un máximo
		A032	El nicho no acota ningún máximo
A003	El individuo posee el fitness menor que los almacenados en el nicho	A013	El nicho acota a un máximo
		A023	El nicho acota a más de un máximo
		A033	El nicho no acota ningún máximo

B El individuo está fuera de los valores de un nicho			
B001	El individuo posee el fitness mayor que los valores del nicho	B011	El nicho acota a un máximo
		B021	El nicho acota a más de un máximo
		B031	El nicho no acota ningún máximo
B002	El individuo posee el fitness mayor de sólo uno de los valores almacenados del nicho.	B012	El nicho acota un máximo
		B022	El nicho acota a más de un máximo
		B032	El nicho no acota ningún máximo
B003	El individuo posee el fitness menor que los almacenados en el nicho		

C El individuo se encuentra entre más de dos nichos	
D El individuo se encuentre entre dos o más nichos	
E El nicho posee en su rango de búsqueda dos o más máximo	
E001	Si los máximos se encuentran a una cierta distancia
E002	Si los máximos se encuentran muy cercano
F Más de un nicho buscan el mismo máximo	
G Los puntos que forman el nicho están alejados	
G001	Los puntos del nicho tienen un fitness elevado
G002	Unos de los puntos del nicho tiene un fitness elevado y el otro un fitness bajo
G003	Los puntos que forman el nicho tienen un fitness bajo.

Se utilizará para explicar cómo funciona la librería en una función sencilla de una variable (función 6.1) y también se desarrollará con una función más complicada dada por (función 6.2)

$$Y = 4 * sen(4 + x)$$

Función 6.1

$$f(x,y) = 3(1-x)^2 e^{-(x^2+(y+1)^2)} - 10(\frac{x}{5} - x^3 - y^5)e^{-(x^2+y^2)} - \frac{1}{3}e^{-((x+1)^2+y^2)}$$

Función 6.2

La representación de la función 6.1 será Figura 6.1

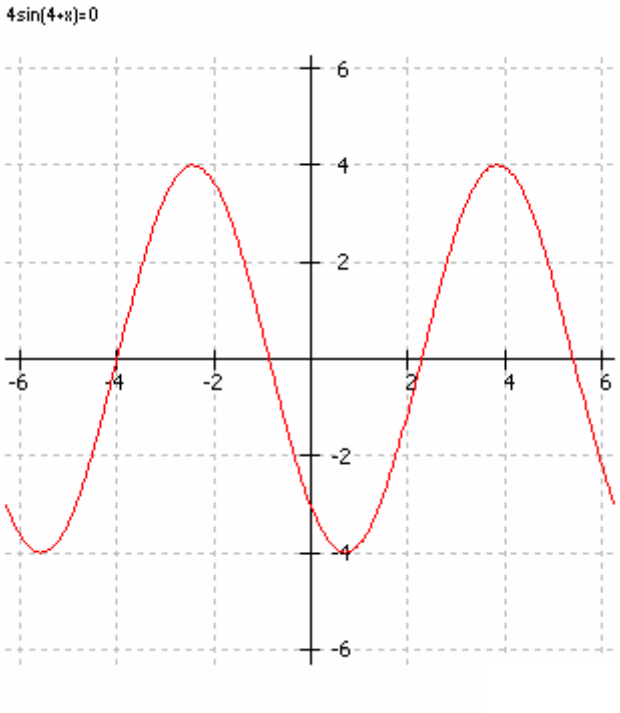


Figura 6.1 Representación Función 6.1

A continuación se muestra una representación de la función 6.2 en la figura 6.2 aunque para ver cómo funciona la librería se usará la representación en plano como la figura 6.3

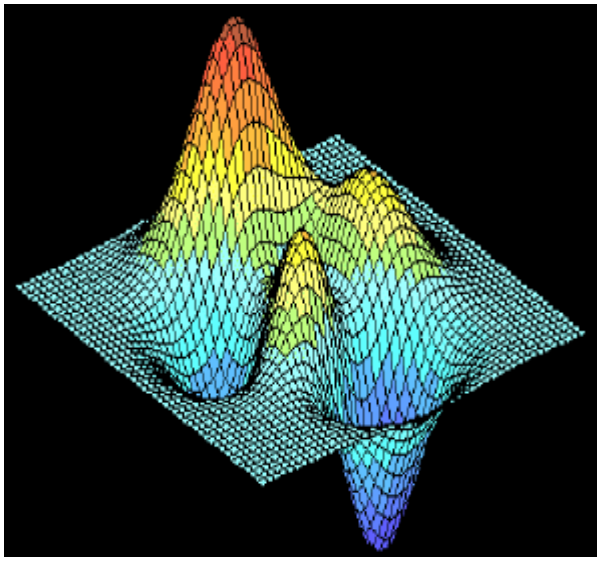


Fig 6.2 Representación de la función 6.2 en 3 dimensiones

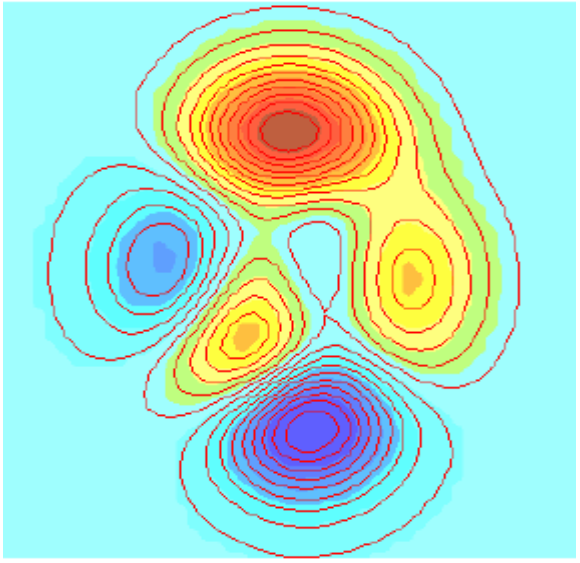


Fig 6.3 Representación de la función 6.2 en 2 dimensiones

A El individuo está entre los valores de un nicho.

En los siguientes casos de usos, el individuo se encuentra entre los valores del nicho.

A001 El individuo posee el fitness mayor que los valores almacenados en el nicho.
Estos casos de usos están formados por un individuo que posee fitness superior a los almacenados en el nicho.

A011 El nicho acota a un máximo. Los valores que forman el nicho tienen delimitada una solución. Además el individuo tiene un valor de fitness mayor que los almacenados estando más próximo a la solución buscada. Entonces se sustituye el menor de los valores almacenados en el nicho por el individuo. De esta forma se van acercando a la solución deseada. Una vez sustituido el valor puede ocurrir que el valor buscado se encuentre entre los acotados por los valores del nicho (casos de uso A), o que la solución buscada no se encuentre acotada por los valores almacenados del nuevo nicho (caso de usos B).

Seguidamente se muestra el ejemplo como el individuo se encuentra entre los valores del nicho figura 6.4 Y como es sustituido uno de los valores del nicho por el individuo figura 6.5

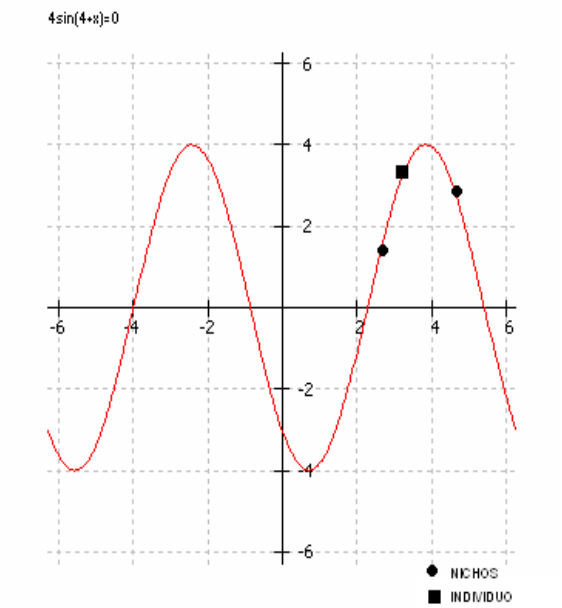


Fig 6.4 Individuo se encuentra valores del nicho y un alto fitness

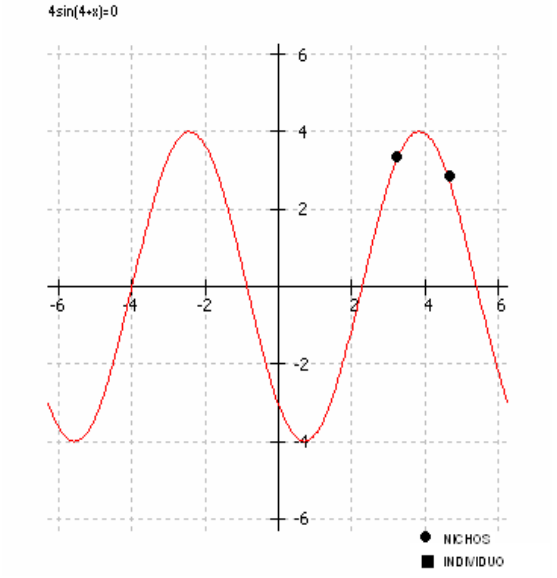


Fig 6.5 Después de aplicar el algoritmo la Fig 6.4

Puede suceder que el máximo se encuentra entre los valores del nicho como muestra en la figura 6.6 y al ejecutarse la librería el máximo ya no se encuentra de delimitado por el nicho como se observa en la figura 6.7.

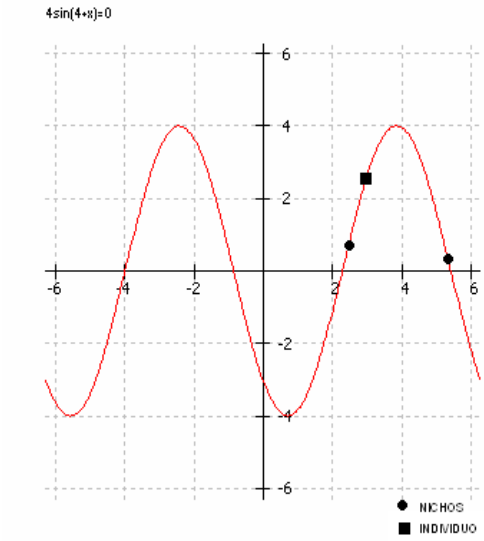


Fig 6.6 Individuo se encuentra valores del nicho

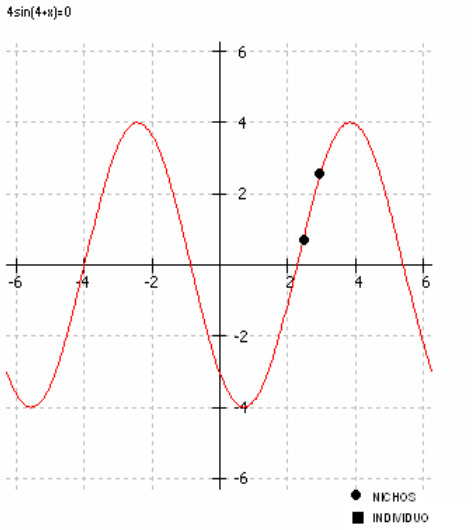


Fig 6.7 Los dos puntos nichos esta misma pendiente

Se mostrará también el caso de uso con un ejemplo de la función 6.2, el individuo se encuentra entre los valores del nicho y posee un fitness elevado. En la Fig 6.8 se muestra como es sustituido unos de los puntos de nicho por el individuo, quedando al final el nicho como se muestra en la Fig 6.9.

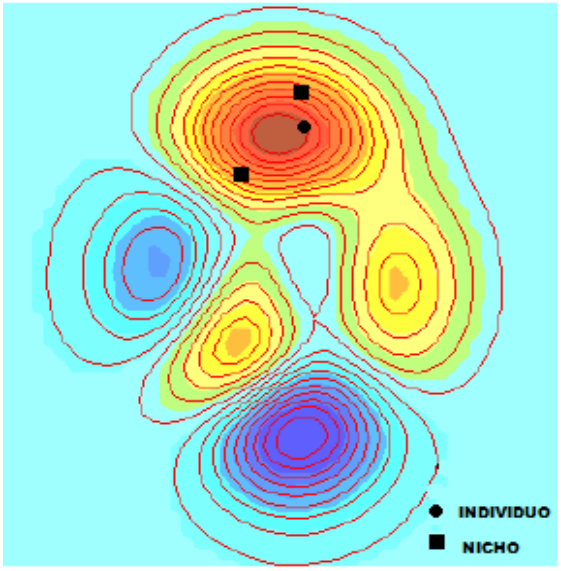


Fig 6.8 Individuo se encuentra valores del nicho

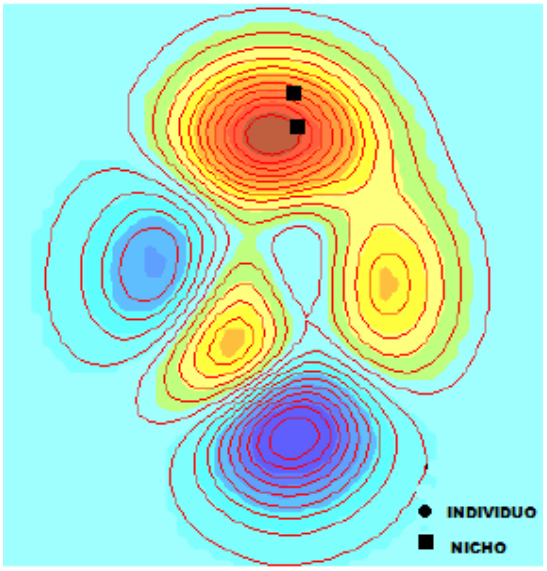


Fig 6.9 Los dos puntos nichos está más cerca una solución

A021 El nicho acota a más de un máximo En este caso el nicho acota a varios máximos y además el individuo tiene un valor de fitness mayor que los del nicho. Como el caso anterior se sustituirá el valor menor del nicho por el individuo. Como norma, un nicho debe acotar o estar cerca sólo de una única solución. En el siguiente ejemplo se muestra como los puntos del nicho contienen dos máximos y el individuo se encuentra entre los dos valores como se muestra en la figura 6.10. A continuación el valor con menor fitness del nicho será sustituido por el individuo quedando como se muestra en la figura 6.11

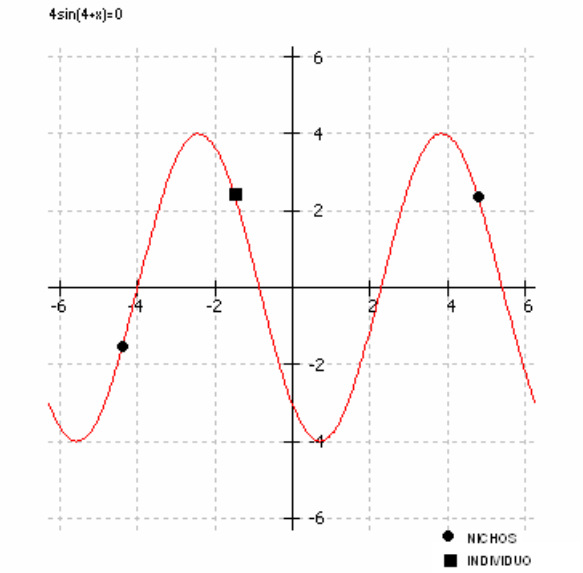


Fig 6.10 Individuo se encuentra valores del nicho

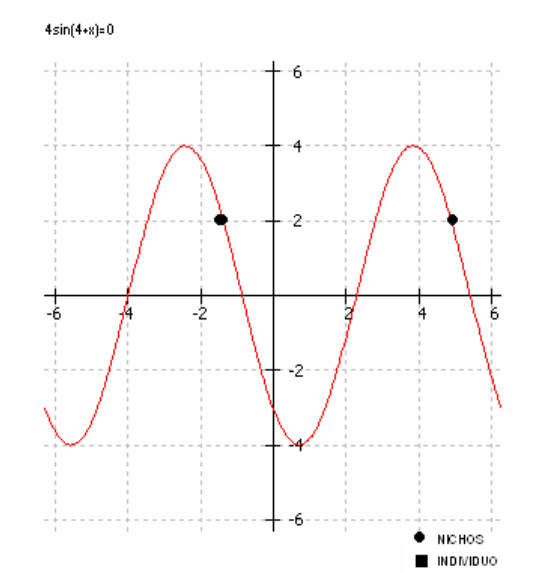


Fig 6.11 El nicho va acotando a una única solución

A continuación, este mismo caso de uso se muestra en la figura 6.12, en la cual se muestra al nicho que contiene dos máximos y el individuo se encuentra entre los dos puntos del nicho. Se muestra también como el punto del nicho con menor fitness es sustituido por el de individuo como se muestra en la figura 6.13

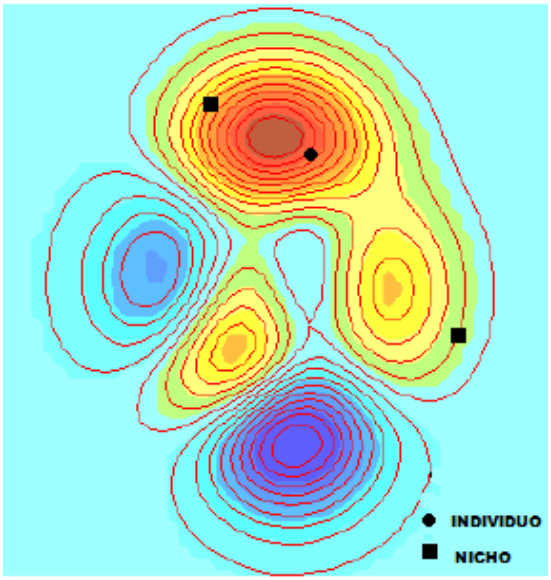


Fig 6.12 Individuo se encuentra valores del nicho

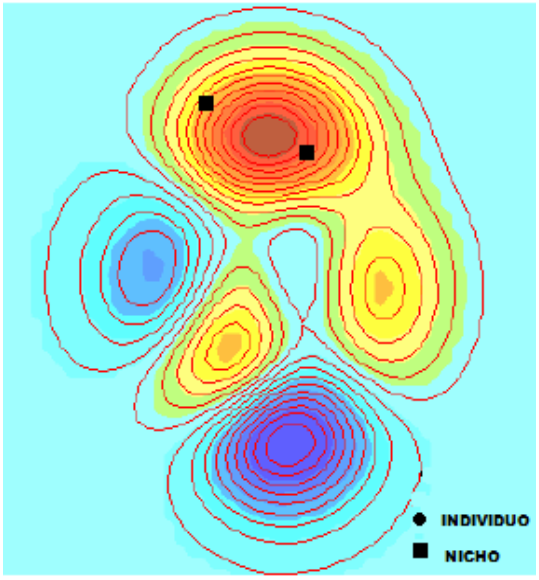


Fig 6.13 El nicho va acotando a una única solución

A031 El nicho no acota ningún máximo. En este caso de uso el nicho no acota a ningún máximo. Como en los casos anteriores el individuo posee un fitness elevado, entonces se sustituye el valor de uno de los nichos por el individuo. De esta forma el nicho se va acercando o acotando a una solución buscada. Como en la figura 6.14 el nicho no contiene ningún máximo y el individuo posee un alto fitness entonces uno de los puntos del nicho será sustituido por el individuo como se muestra en la figura 6.15

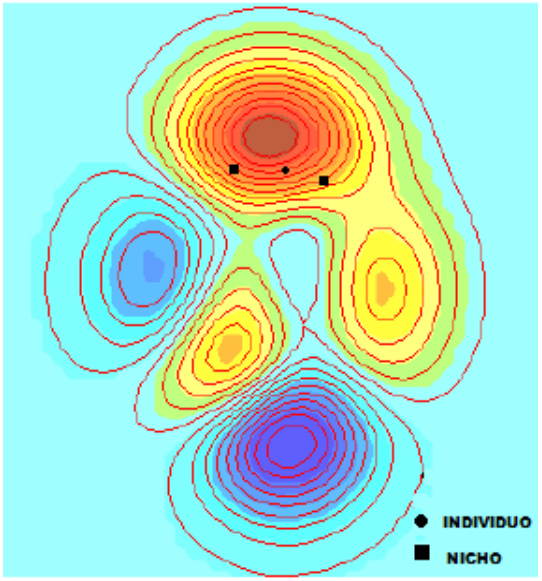


Fig 6.14 El individuo posee fitness alto

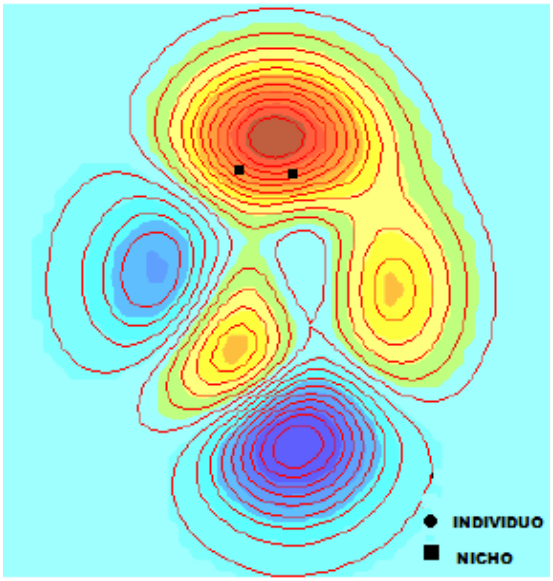


Fig 6.15 El nicho se va acercando a la solución

A002 El individuo posee el fitness mayor de sólo uno de los valores almacenados del nicho.

El individuo se encuentra entre los valores del nicho y es superior sólo a uno de los valores del nicho.

A012 El nicho acota a un máximo. Este caso tiene la misma solución que el caso A011. Se sustituye el valor del nicho inferior por el individuo y como en el caso antes citado puede dar como solución que el máximo se encuentre entre los valores del nicho (caso uso A) o que se encuentre fuera de los valores del nicho (caso de usos B).

En el ejemplo, el individuo está entre los valores del nicho como se muestra en la figura 6.16, la librería modifica el nicho como la figura 6.17 acercándose más a la solución.

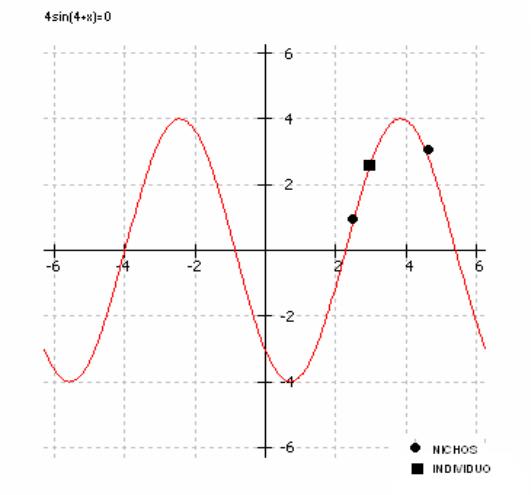


Fig 6.16 El individuo se encuentre entre valores nicho

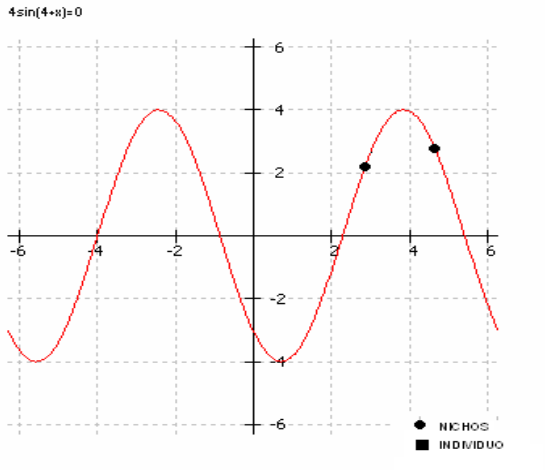


Fig 6.17 El nicho se va acercando a la solución

Este mismo caso puede aplicarse a la figura 6.18 y como es sustituido uno de los puntos que forman el nicho por el individuo dando como resultado la figura 6.19.

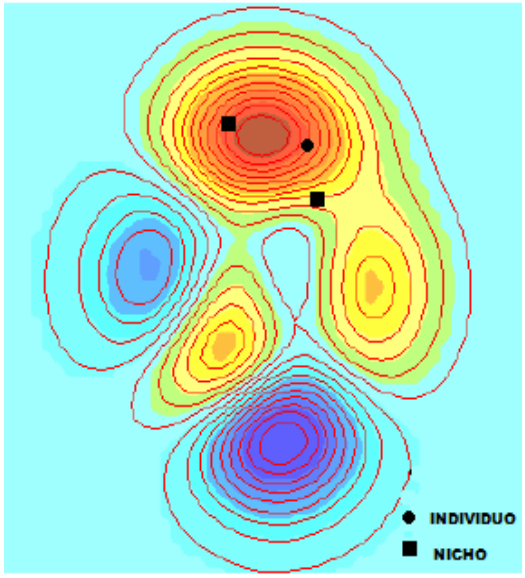


Fig 6.18 El individuo se encuentre entre los valores nicho

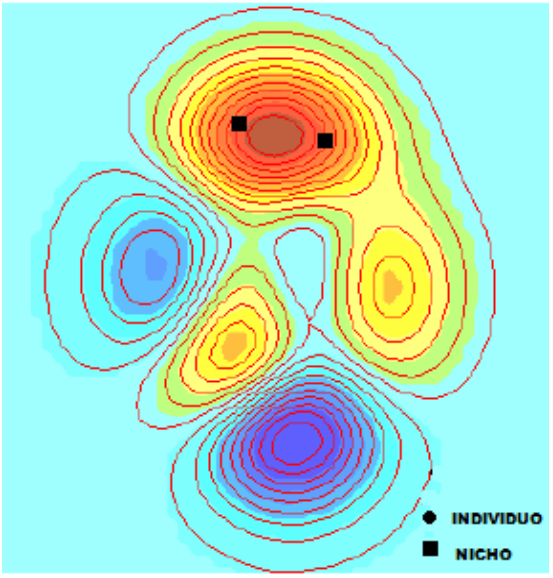


Fig 6.19 El nicho se va acercando a la solución

A022 El nicho acota a más de un máximo. Este caso es muy parecido al caso anterior, la única diferencia que el nicho está acotando a dos máximos como se muestra en la figura 6.20. Y la resolución es idéntica como se muestra la figura 6.21, el individuo sustituye uno de los valores del nicho acercándose a una solución.

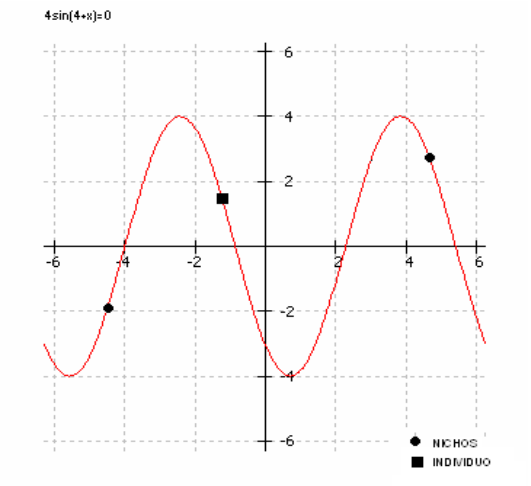


Fig 6.20 El individuo se encuentre entre valores nicho

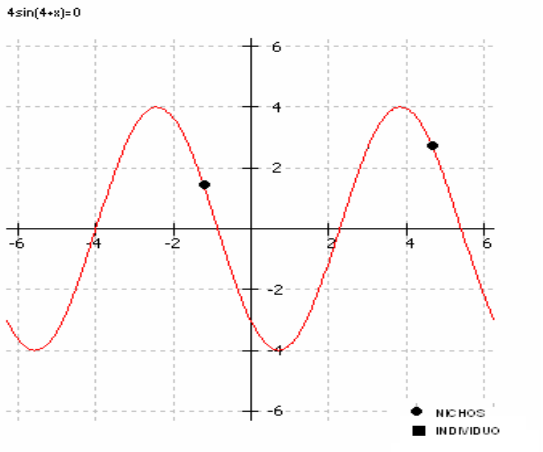


Fig 6.21 El nicho se va acercando a una solución

Este mismo caso de uso se puede observar en la figura 6.22.Uno de los valores del nicho es sustituido por el individuo como se muestra en la figura 6.23.

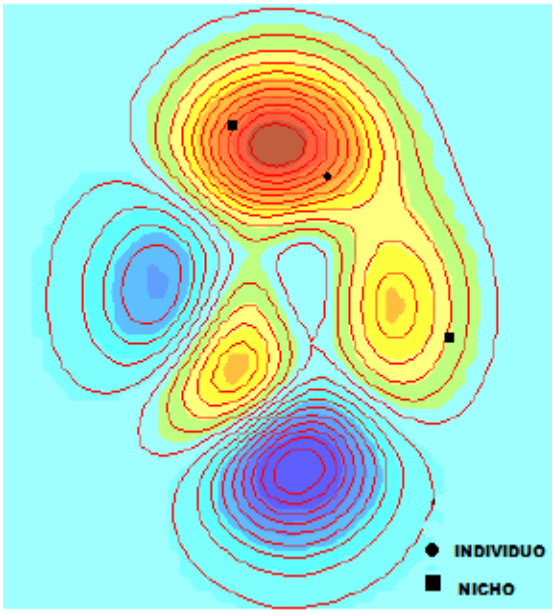


Fig 6.22 El individuo se encuentre entre valores nicho

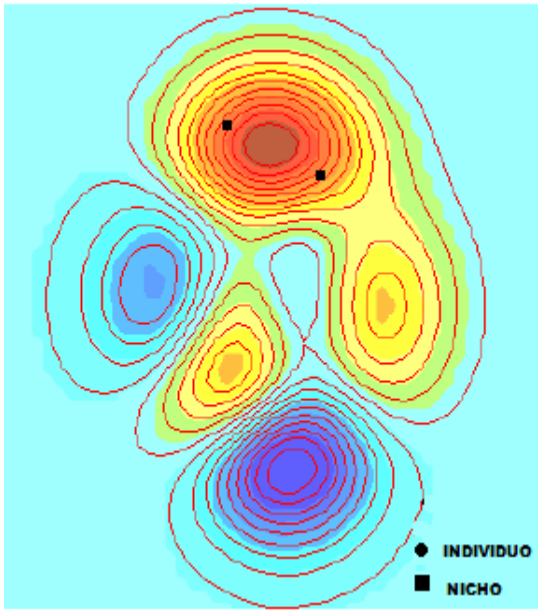


Fig 6.23 El nicho se va acercando a la solución

A032 El nicho no acota ningún máximo. En este caso de uso el nicho no acota a ningún máximo y el individuo posee un fitness superior a uno de los valores del nicho. La librería sustituye el valor inferior de los nichos por el individuo. De esta forma el nicho se va acercando o acotando a una solución buscada. En este caso no quedará la solución acotada por el nicho pero se va acercando los valores del nicho a la solución buscada. Un ejemplo sería el de la figura 6.24 en la que se puede comprobar que el nicho no contiene ningún máximo y el individuo se encuentra dentro de su rango. Y como se obtiene como resultado la figura 6.25, siguiendo sin acotar ningún máximo pero el nicho se va acercado a una solución

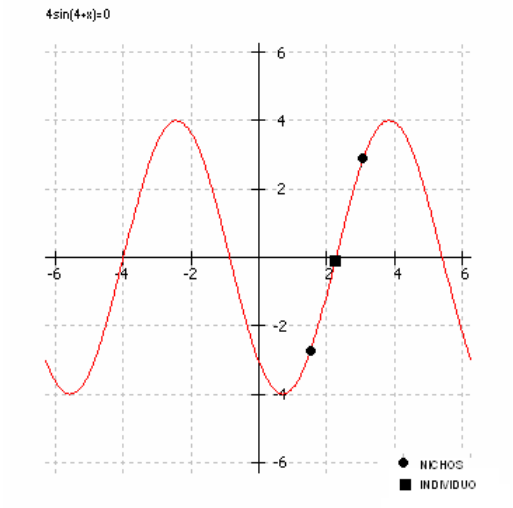


Fig 6.24 El nicho no contiene ningún máximo

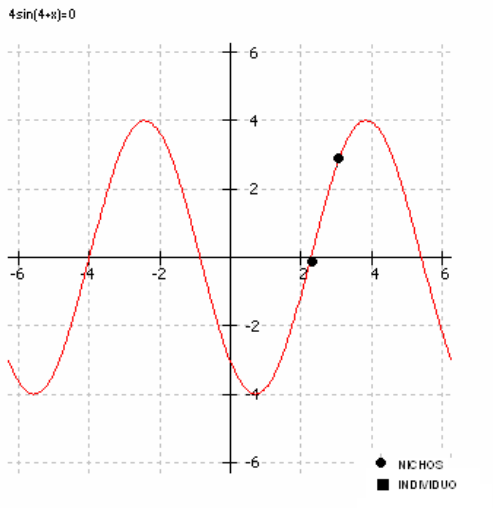


Fig 6.25 El nicho se va acercando a la solución

También se puede comprobar este caso de uso en la figura 6.26. El nicho no contiene ningún máximo y el individuo se encuentra entre los valores del nicho. Y como el valor de un de nicho es sustituido por el individuo acercándose más al máximo como se muestra en la figura 6.27.

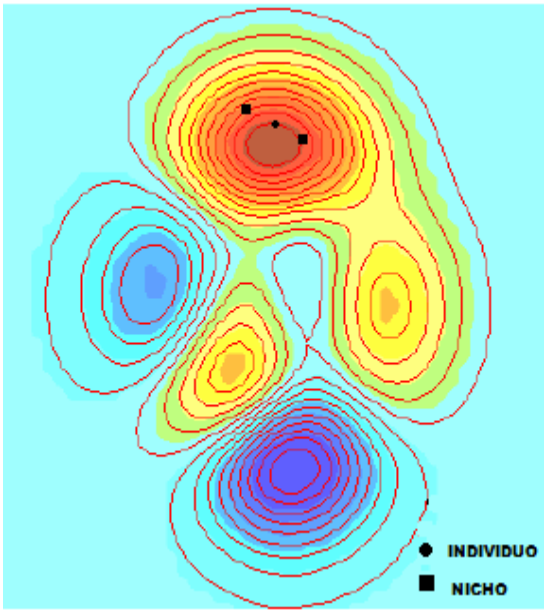


Fig 6.26 El nicho no contiene ningún máximo

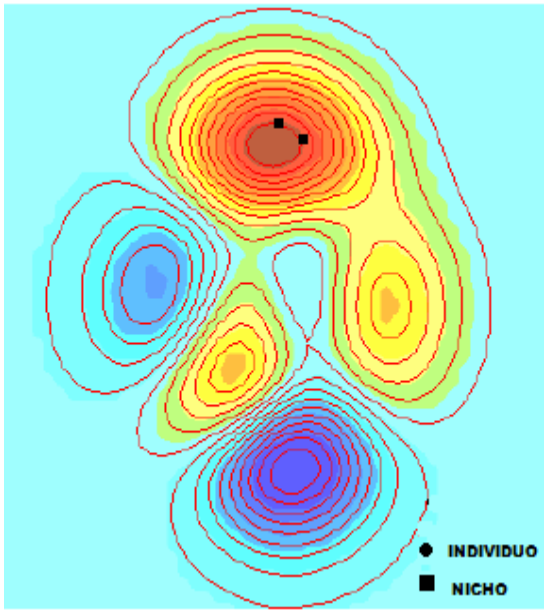


Fig 6.27 El nicho se va acercando a una solución

A003 El individuo posee el fitness menor que los almacenados en el nicho. En estos casos de uso el individuo que se va a tratar tiene un fitness menor que los almacenados que forman el nicho y se encuentra entre los valores que acota el nicho.

A013 El nicho acota a un máximo. En este caso de uso el nicho acota la solución de un máximo y el individuo posee un fitness menor que los almacenados en el nicho. Es posible que el nicho delimite un máximo y un mínimo. Este individuo no será tratado y no modificara ningún valor del nicho al poseer un fitness inferior. Un ejemplo se puede observar en figura 6.28 y figura 6.29. El individuo posee un fitness menor que valores guardados por el nicho, y sabe que existe un mínimo y un máximo pero con la información que posee no modifica ningún valor del nicho, a no aportar ninguna información para acercarse a una solución.

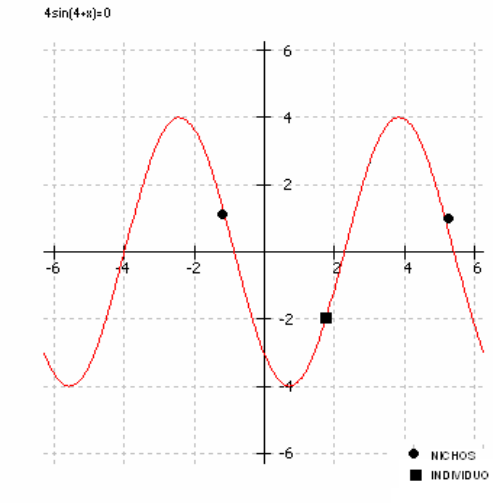


Fig 6.28 El nicho contiene máximo y un mínimo

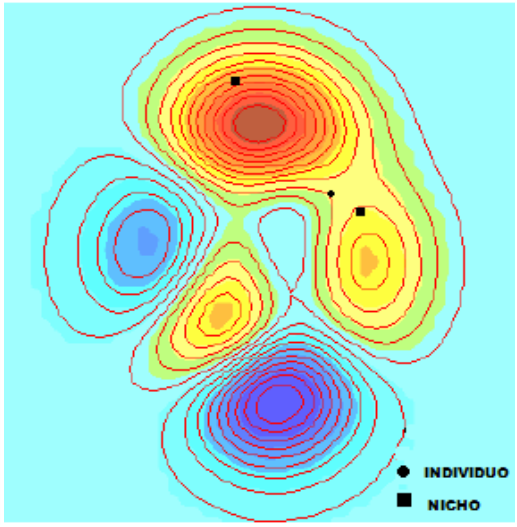


Fig 6.29 El nicho contiene máximo y un mínimo

A023 El nicho acota a más de un máximo. En este caso el nicho acota más de una solución y el individuo posee un fitness menor. Como en el caso de uso anterior el individuo no será tratado ni tampoco modificará los valores del nicho. Se muestra en la figura 6.30 y la figura 6.31

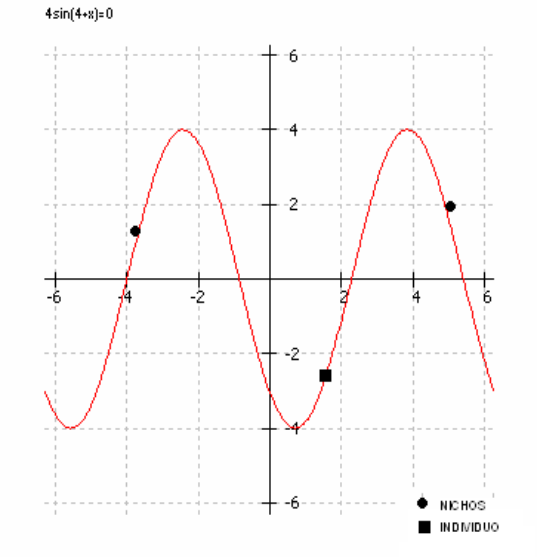


Fig 6.30 El nicho contiene más de un máximo

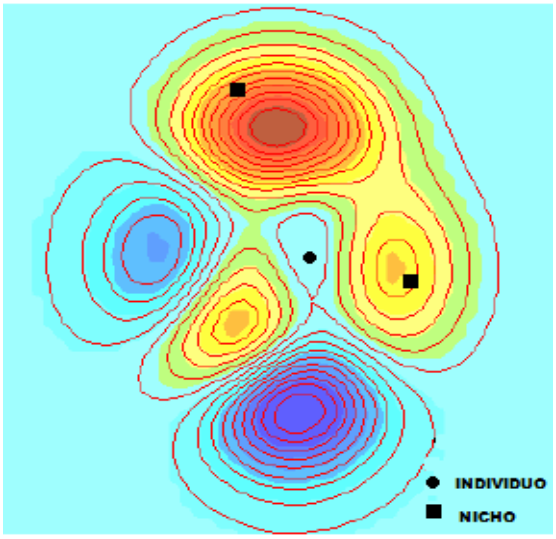


Fig 6.31 El nicho contiene más de un máximo

A033 El nicho no acota ningún máximo El valor de individuo es inferior a los valores almacenados en el nicho. No modificándose ningún valor del nicho como aparece en la figura 6.32 y la figura 6.33

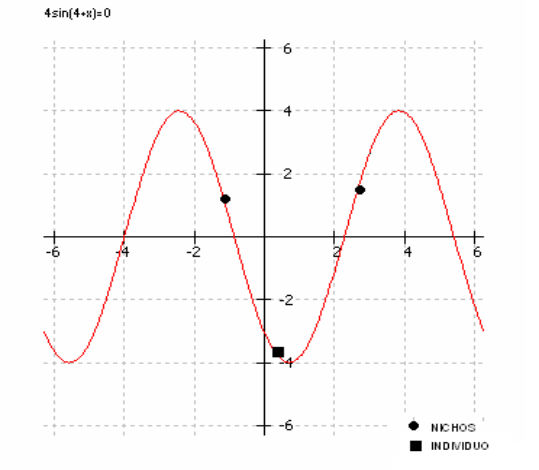


Fig 6.32 El nicho no es modificado

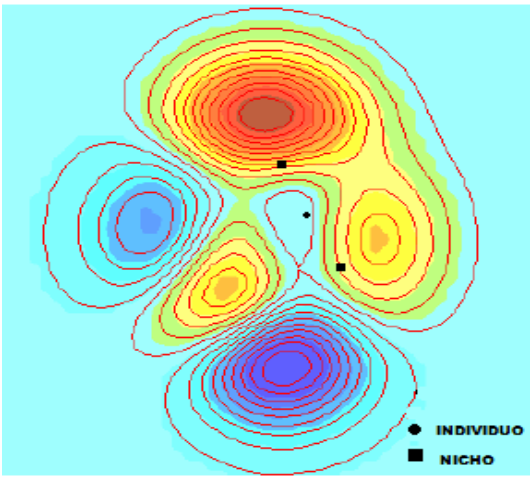


Fig 6.33 El nicho no contiene un máximo

B El individuo está fuera los valores del nicho.

Ahora se estudiarán los casos de usos cuando el individuo que se está tratando se encuentra fuera de los valores acotados por el nicho

B001 El individuo posee el fitness mayor que los valores del nicho. El individuo que se va trata posee un fitness mayor que los valores del nicho y se distinguirán los siguientes casos:

B011 El nicho acota a un máximo. El individuo posee un nicho elevado pero está fuera del rango del nicho. Será sustituido el valor del nicho con menor fitness por el individuo. Esto puede traer como consecuencia que el nicho que antes tenía como solución de búsqueda un solo máximo pueda tener varios o que el máximo que estaba acotado deje de estarlo. En el ejemplo se muestra como el individuo posee un fitness elevado y está fuera del nicho como muestra la figura 6.34. En la figura 6.35 se muestra cómo se modifica el nicho; ahora el máximo no se encuentra acotado pero ha mejorado uno de los valores del nicho.

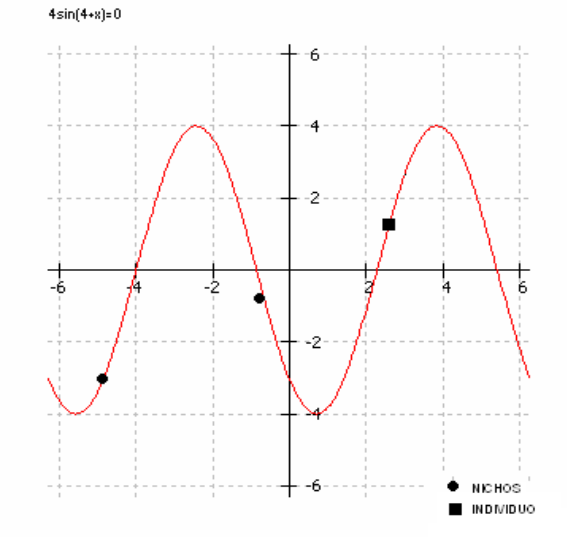


Fig 6. 34 El nicho contiene un máximo pero poseen fitness bajo

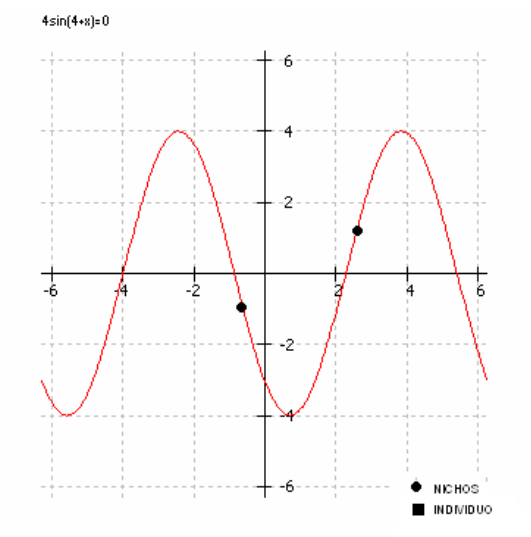


Fig 6.35 El nicho se va acercando a una única solución

Otro ejemplo que cumple este caso de uso será el de la figura 6.36; el máximo está acotado por los valores del nicho y el individuo posee un fitness elevado. Se produce el cambio de sustituir el peor punto que forma el nicho por el individuo como se muestra en la figura 6.37; se puede ver cómo el nicho se va acercando a una solución.

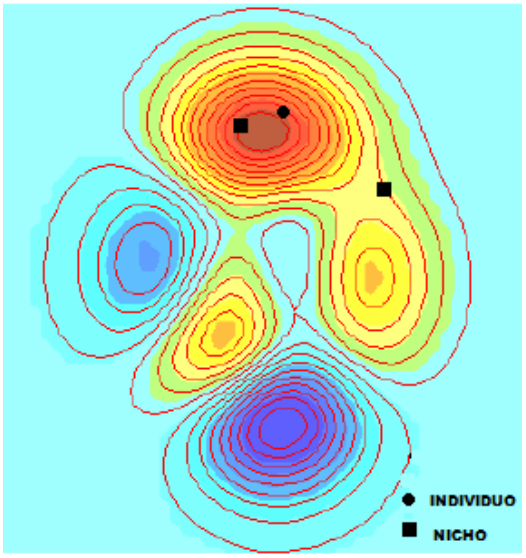


Fig 6.36 El nicho tiene acotado una solución

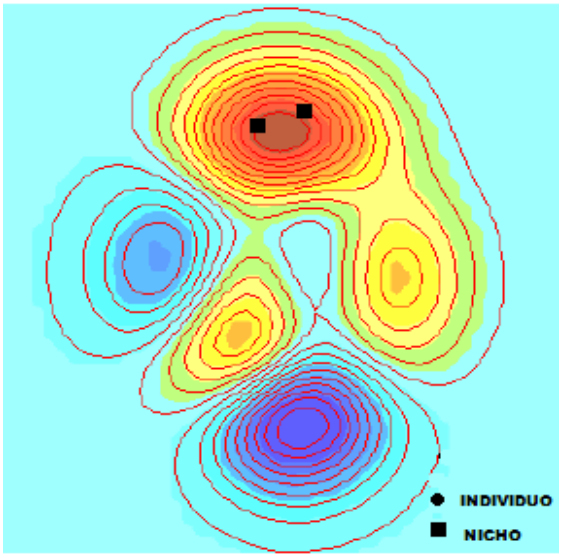


Fig 6.37 El nicho no acota la solución pero está más cerca

B021 El nicho acota a más de un máximo. Es idéntico que el caso anterior, lo único que en este caso el nicho acota a más de un máximo. En el ejemplo de la figura 6.38 se puede observar que el nicho acota dos soluciones correctas y como el individuo se encuentra fuera del rango del nicho. Como el individuo sustituye uno de los valores del nicho se puede observar en la figura 6.39. Este caso no debería darse que aumente el número de soluciones buscadas en el nicho.

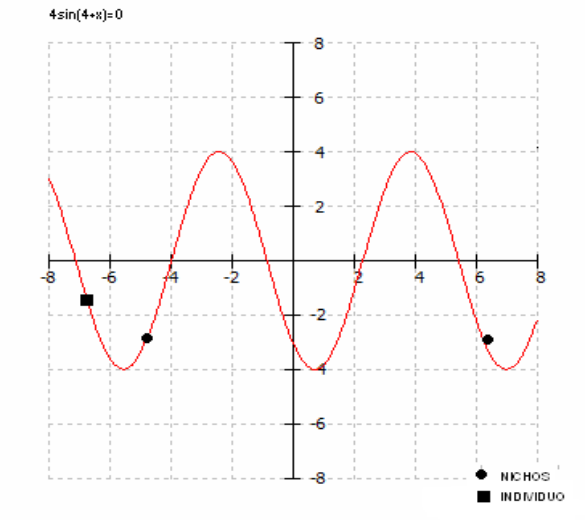


Fig 6.38 El nicho contiene más de un máximo

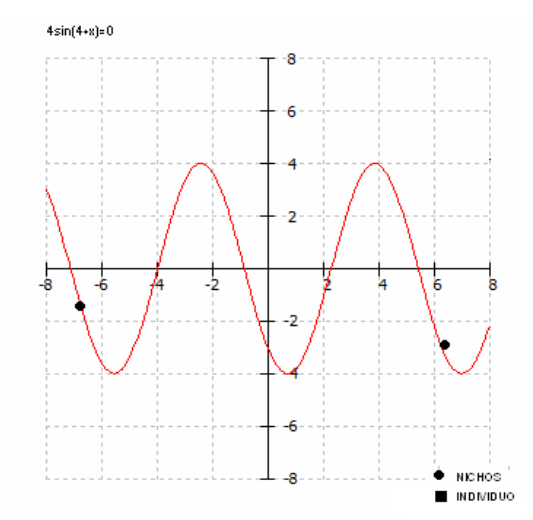


Fig 6.39 El nicho va aumentando su rango

En el ejemplo de la figura 6.40 puede mejorar el nicho, al producirse el cambio se mejora la solución al acercarse más a un máximo como se muestra en la figura 6.41

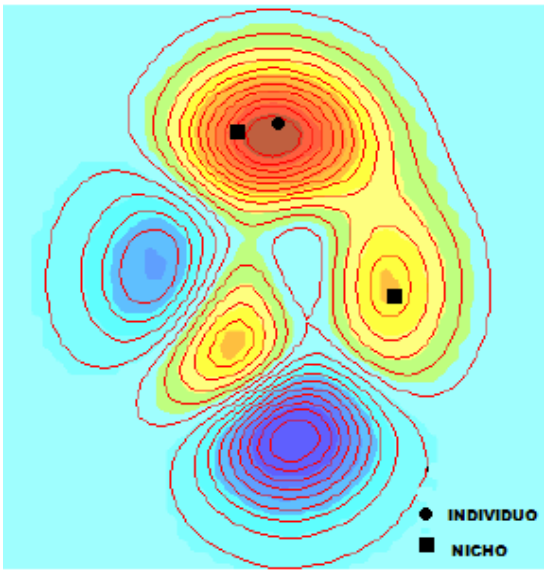


Fig 6.40 El nicho contiene más de un máximo

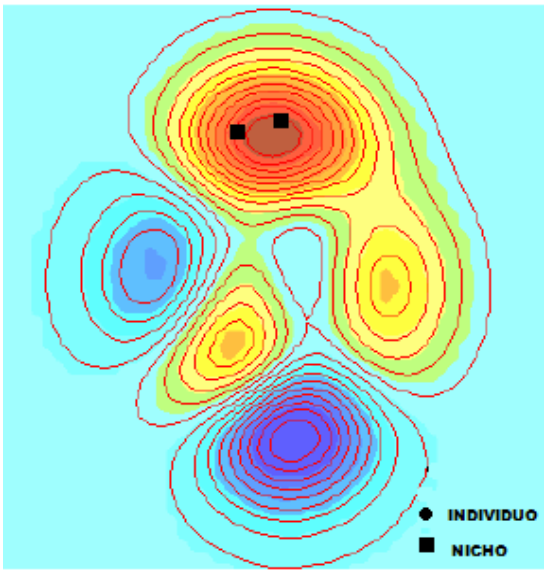


Fig 6.41 El nicho se ha acercado a un máximo

B031 El nicho no acota ningún máximo. Aquí el individuo posee un fitness mayor que los valores del nicho. El valor menor del nicho será sustituido por el individuo. Al producirse este cambio la solución puede que esté acotada por los valores del nicho. O no quede delimitado, pero estará más cerca a la solución buscada. En la figura 6.42 se puede observar que los valores del nicho tienen un fitness inferior al individuo y es modificado acercándose al máximo como se muestra en la figura 6.43

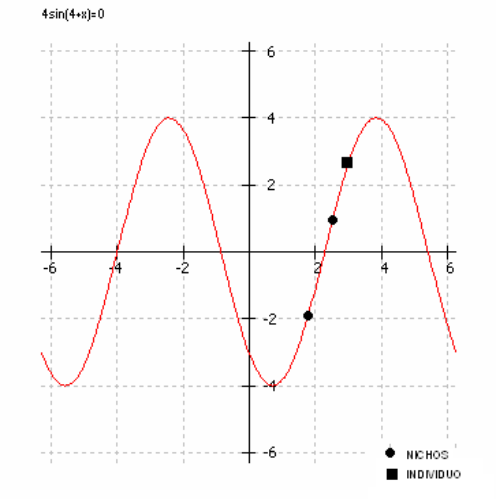


Fig 6.42 El nicho no contiene ningún máximo

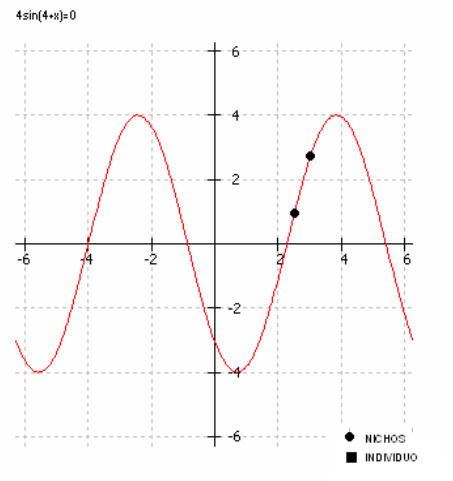


Fig 6.43 El nicho se va acercando a una solución

En el siguiente ejemplo también el individuo posee un fitness más elevado que el nicho como se muestra en la figura 6.44 y es sustituido uno de los valores del nicho por el individuo (figura 6.45) acercándose ese nicho a una solución deseada

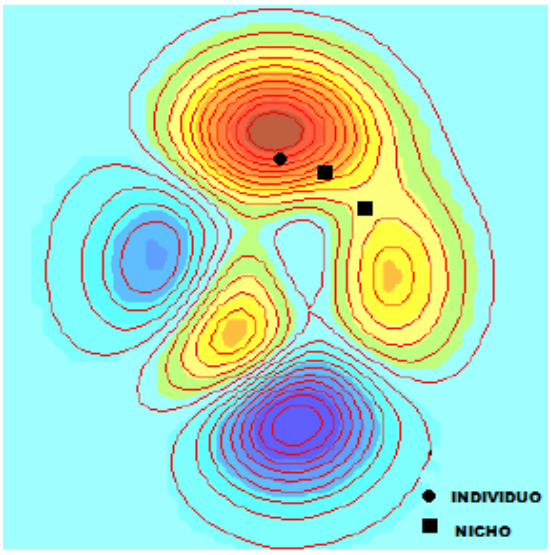


Fig 6.44 El nicho no contiene ningún máximo

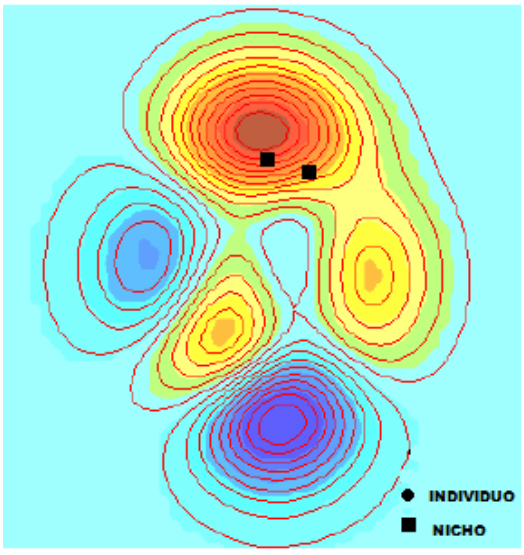


Fig 6.45 El nicho se va acercando a una solución

B002 El individuo posee el fitness mayor de sólo uno de los valores almacenados del nicho. El individuo se encuentra fuera del rango del nicho almacenado y posee un fitness mayor que uno de los puntos que forman el nicho

B012 El nicho acota a un máximo. El máximo se encuentra entre el rango que forma el nicho como se muestra en la figura 6.46 y el individuo sustituirá el valor del nicho como se muestra en la figura 6.47. Aumentado el rango del nicho y acercándose también a la solución de otro máximo.

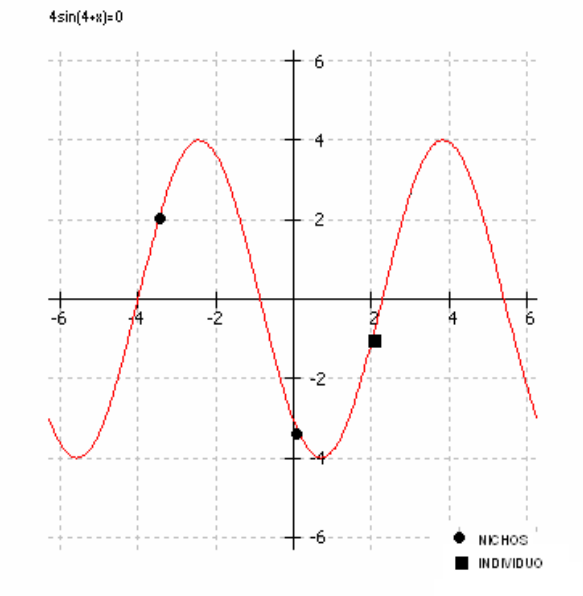


Fig 6.46 El nicho contiene máximo

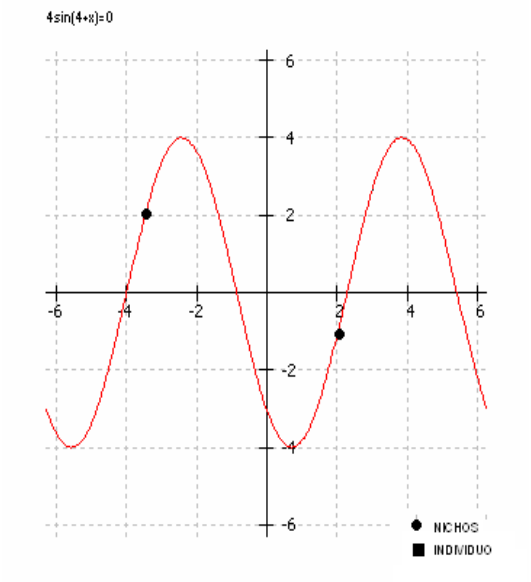


Fig 6.47 El nicho aumenta el acotamiento

La figura 6.48 explica como el nicho se encuentra entre un máximo y como el individuo se encuentra fuera del rango pero posee un fitness mayor que uno de los puntos que forma el nicho. La figura 6.49 muestra como el punto del nicho con menor fitness es sustituido por el individuo acercándose el nicho a una solución.

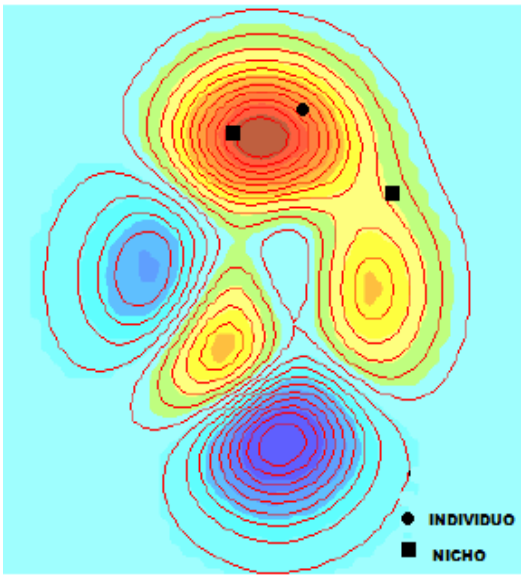


Fig 6.48 El nicho contiene máximo

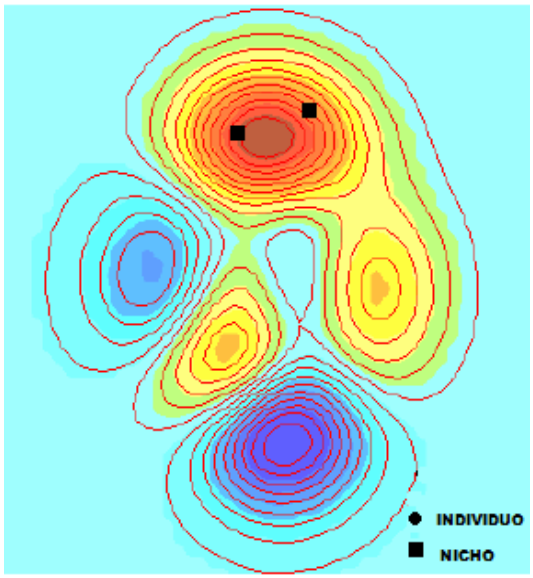


Fig 6.49 El nicho se va acercando a un máximo

B022 El nicho acota a más de un máximo: Este caso es muy parecido al anterior lo único que el nicho acota a varios máximos como en la figura 6.50. Y al cambiar el nicho puede que aumenta las distancias entre los puntos del nicho como se puede observar en la figura 6.51.

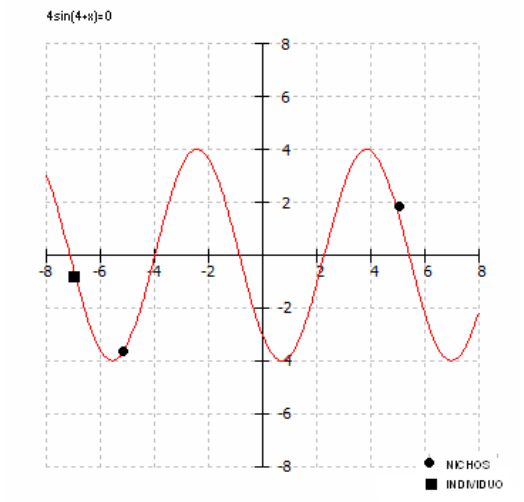


Fig 6.50 El nicho contiene varios máximo

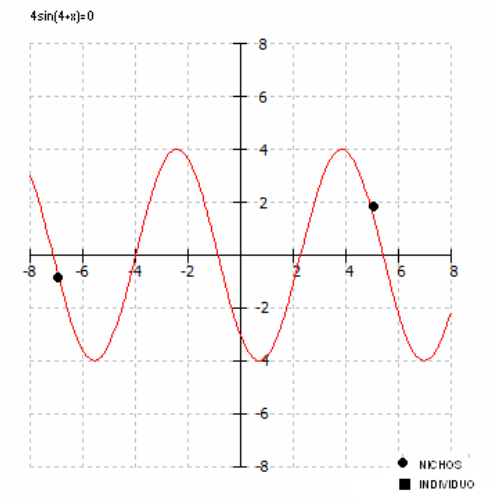


Fig 6.51 El nicho amplio su acotamiento del máximo

En la figura 6.52 se observa como el nicho acota a varios máximos y como el individuo al poseer un fitness mayor que uno de los componentes del nicho, modificará el nicho acercándose a una de las soluciones al problema como se muestra en la figura 6.53.

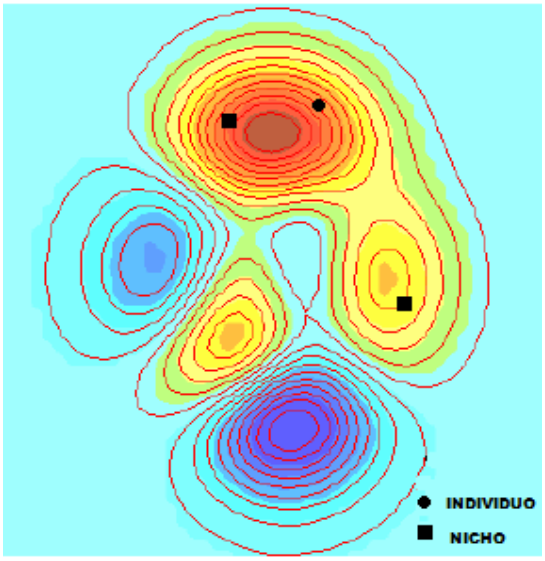


Fig 6.52 El nicho contiene varios máximo

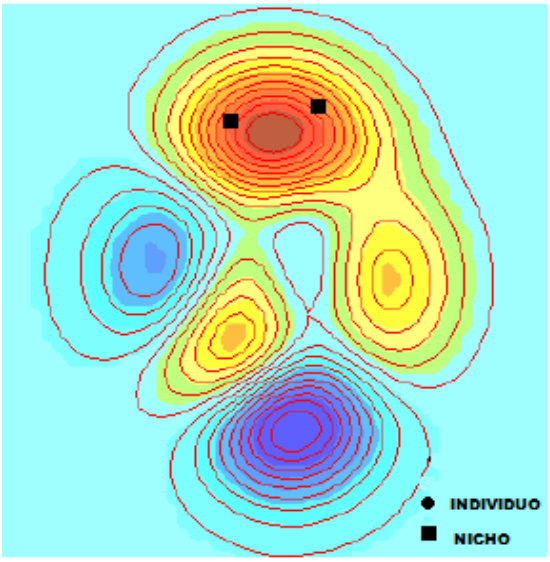


Fig 6.53 El nicho se va acercando a una solución

B032 El nicho no acota ningún máximo: En este caso de uso el nicho no acota ningún máximo y el individuo posee un fitness mayor que unos de los puntos almacenados en nicho. Entonces el punto del nicho con menor fitness será sustituido por el individuo, de esta forma se acerca más el nicho a la solución buscada, e incluso puede ocurrir que la solución buscada ahora se encuentre dentro del rango del nicho. En el siguiente ejemplo figura 6.54 y la figura 6.55 el nicho amplía su rango de búsqueda.

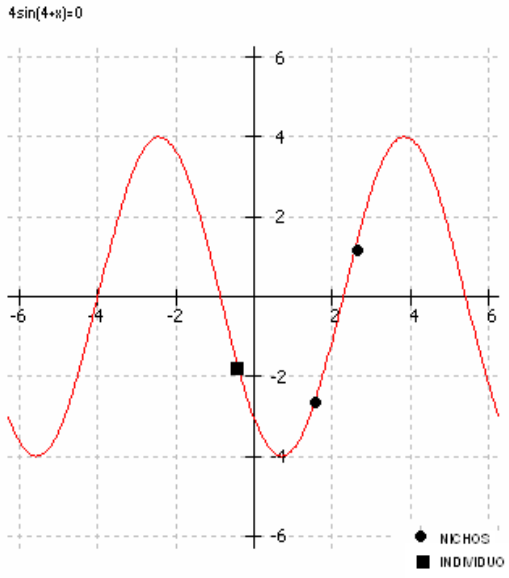


Fig 6.54 El nicho no contiene máximo

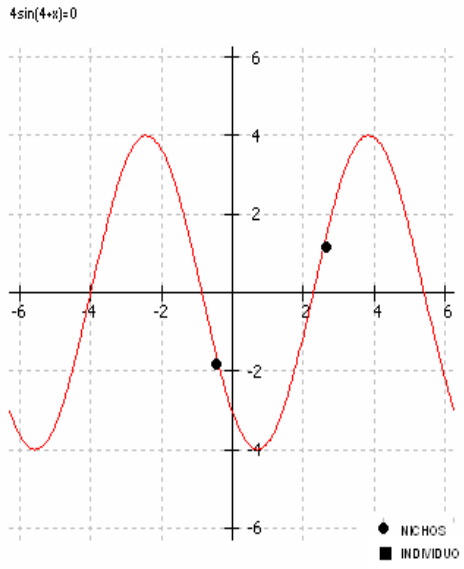


Fig 6.55 El nicho amplio su acotamiento del máximo

Otro ejemplo sería la imagen 6.56 y la imagen 6.57 correspondiendo al mismo caso de uso pero este caso mejora el nicho al acercarse más a un máximo e incluso lo acota.

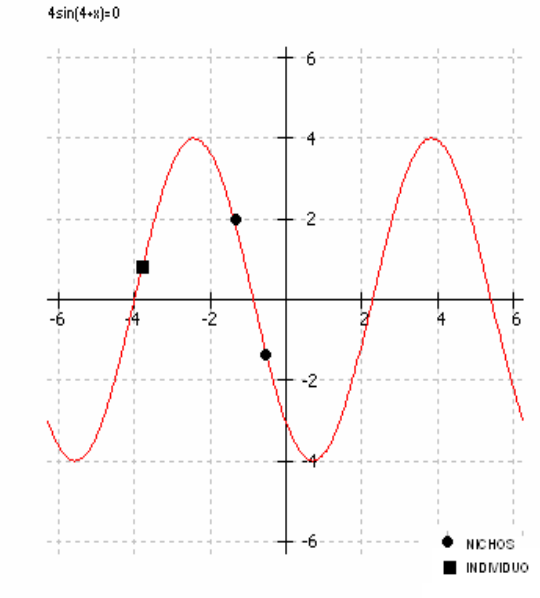


Fig 6.56 El nicho no contiene máximo

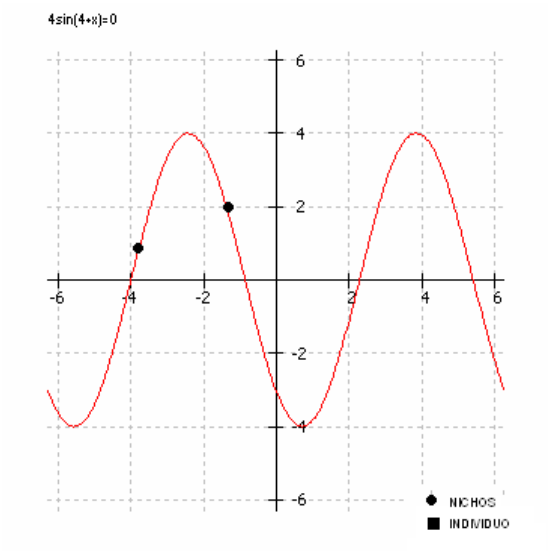


Fig 6.57 El nicho mejorado y acotado la solución

Para finalizar este caso de uso, se mostrará el ejemplo del nicho que no contiene ningún máximo como se muestra en la figura 6.58 y cómo es sustituido uno de los valores del nicho por el individuo como se muestra en la figura 6.59. Se observa que se va acercando a una posible solución

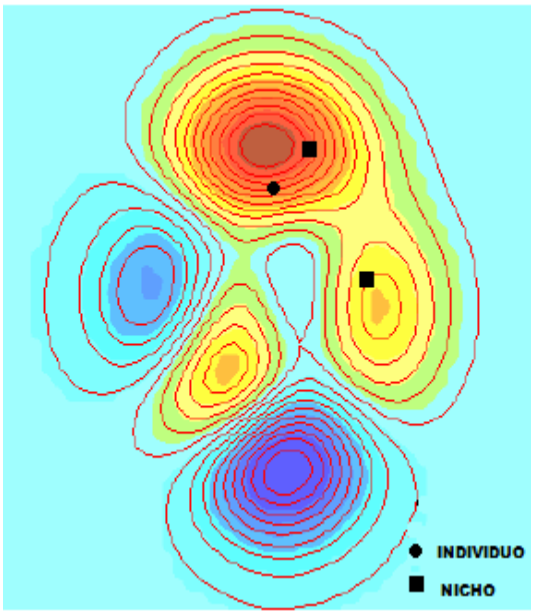


Fig 6.58 El nicho no contiene máximo

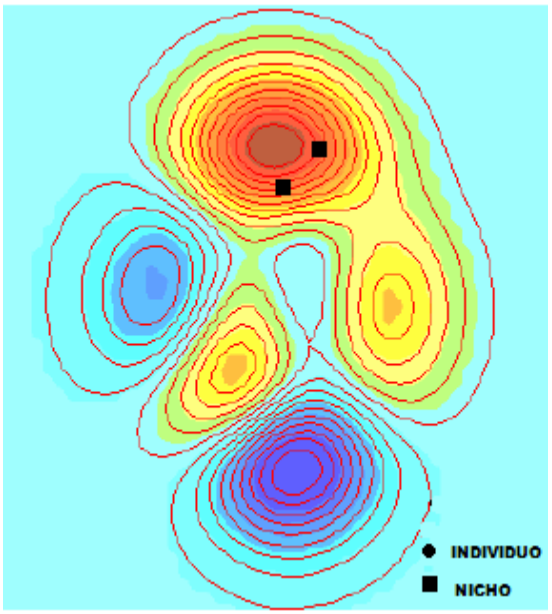


Fig 6.59 El nicho mejorado y se acerca a la solución

B003 El individuo posee un fitness menor de los almacenados en el nicho: En este caso el individuo no aporta ninguna información válida al poseer un fitness menor de lo que están almacenados en el nicho, no siendo útil para obtener ninguna solución. Como se muestra en la figura 6.60 y la 6.61, no se modificará el nicho.

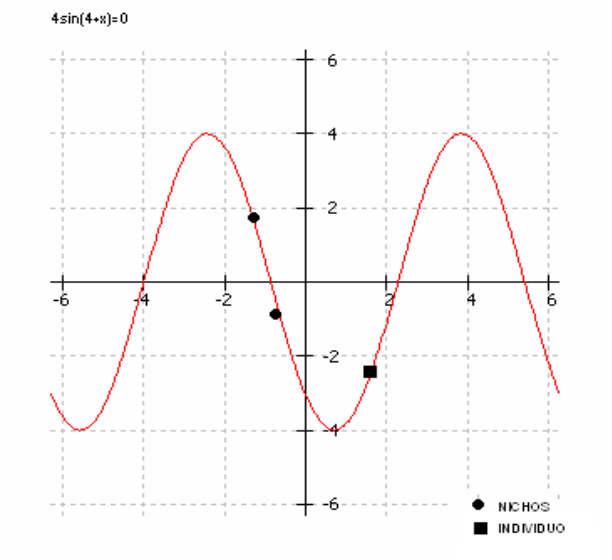


Fig 6.60 El individuo posee un fitness menor que el nicho

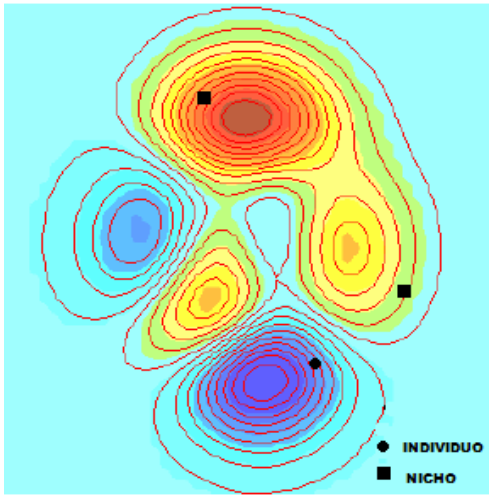


Fig 6. 61 El individuo no aporta información para mejorar nicho

C El individuo se encuentra entre más de dos. En este caso de uso el individuo que se va a tratar se encuentra entre dos o más nichos. Para resolver se busca el nicho más cercano al individuo. Una vez localizado se realizará el mismo proceso como un individuo que está fuera de los valores de un nicho. (Casos de uso B). Se puede observar en la figura 6.62 como existen dos nichos y un individuo. Se buscará aquel de menor distancia entre los nichos y el individuo. Como se puede observar el individuo está más cerca del nicho B y realizará los cambios con ese nicho.

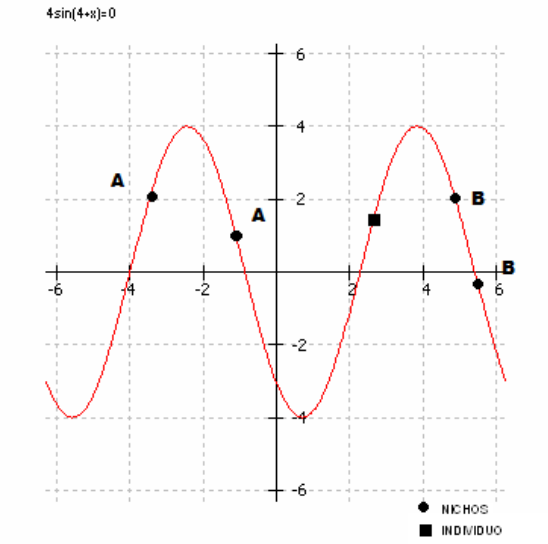


Fig 6.62 Existe dos nichos y un individuo

Este mismo caso de uso se puede aplicar en la figura 6.63. Se observa que existen dos nichos y un individuo, y cómo el individuo interactúa con el nicho más cerca (nicho B) y cómo es modificado por el individuo, como se muestra en la figura 6.64

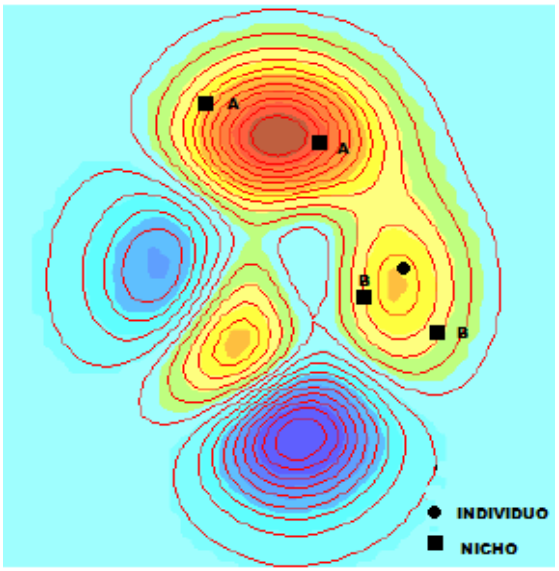


Fig 6.63 Existe dos nichos y un individuo

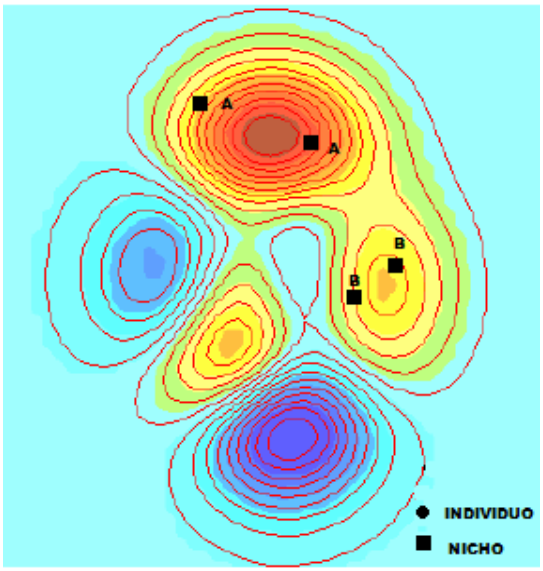


Fig 6.64 El individuo modifica los valores nicho más cercano

D El individuo se encuentre entre dos o más nichos. Como en caso anterior se busca el nicho más cercano al individuo y una vez encontrado se tratará el individuo con su nicho más próximo (sería los casos de uso de tipo A). Una prueba se muestra en la figura 6.65 donde existen dos nichos y un individuo. Entonces el algoritmo busca aquellos puntos del nicho que estén más cerca de él (en el ejemplo sería el nicho del B)

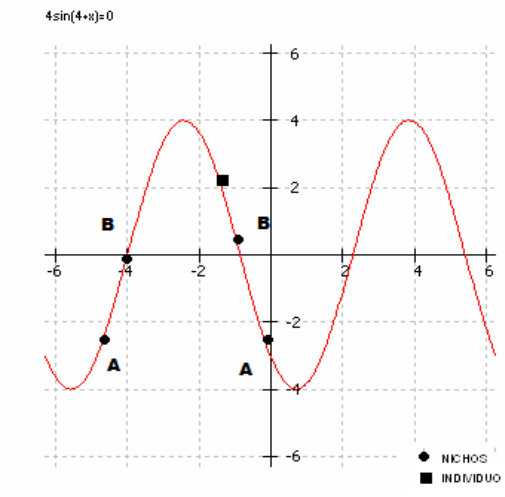


Fig 6.65 Existe dos nichos y un individuo

Otro ejemplo se puede observar la figura 6.66, donde el individuo se encuentra entre dos nichos A y B , pero sólo actuará y modificará al nicho más cercano al individuo que en este caso sería el A.

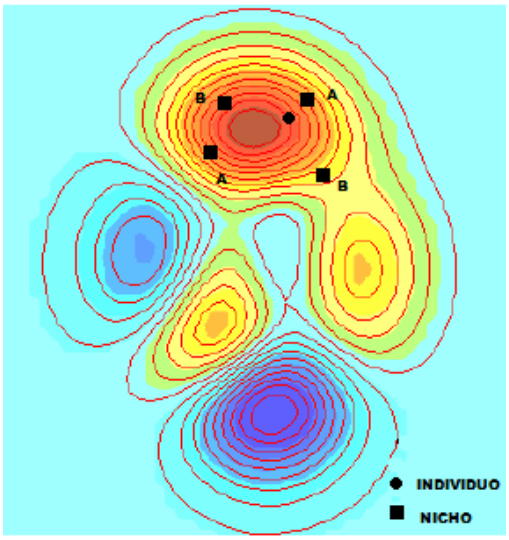


Fig 6.66 Existe dos nichos y un individuo

E El nicho posee en su rango de búsqueda dos o más máximo. El nicho acota a más de una solución siendo incorrecto porque cada nicho debería buscar solo una única solución.

E001 Si los máximos se encuentran a una cierta distancia. Si los máximos se encuentran a una cierta distancia, y si es superior a una constante que se denomina radio de nicho se dividirá el nicho en dos nichos y cada uno tendrá el valor de unos de los nichos y el otro punto será un valor cercano al punto del nicho. Se puede observar el ejemplo en figura 6.67, en el que existe un nicho que está a una cierta distancia y que posee un alto fitness. El algoritmo parte ese nicho en dos y para crear los siguientes puntos de los nichos coge un punto aleatorio cercano al nicho

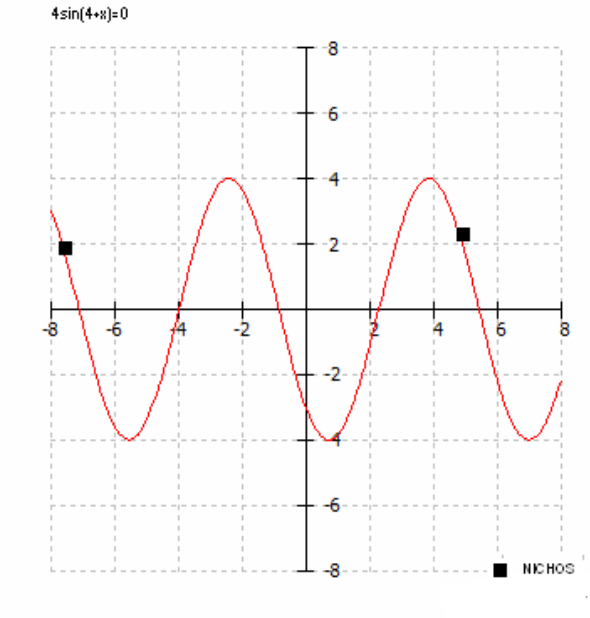


Fig 6.67 Los puntos del nichos esta alejados y con alto fitness los dos

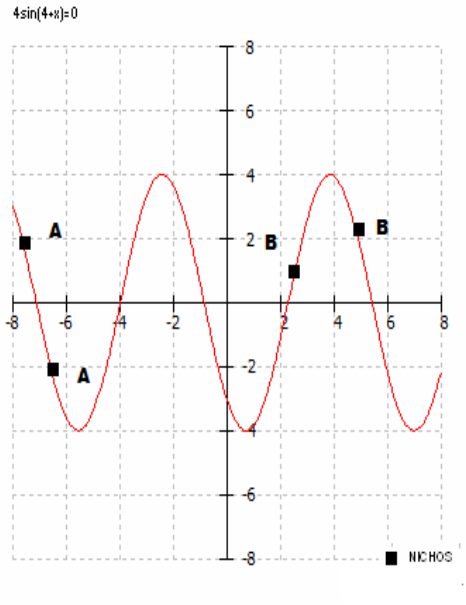


Fig 6.68 El nicho se divide en dos

E002 Si los máximos se encuentran muy cercanos. Si existen máximos que son tan cercanos y las distancias que existen entre ellos es inferior al radio de nicho, es bastante difícil que la librería pueda distinguir ese dos máximo, al considerar que los valores que se obtengan pertenecen al mismo nicho. Hay que estudiar el radio que debe utilizarse; dependiendo del problema se aumentará o disminuirá. A continuación se muestra una función que tienes los máximos muy cercanos (figura 6.69). Dependiendo del radio elegido ese nicho pueda ser dividido en otro dos nicho o no.

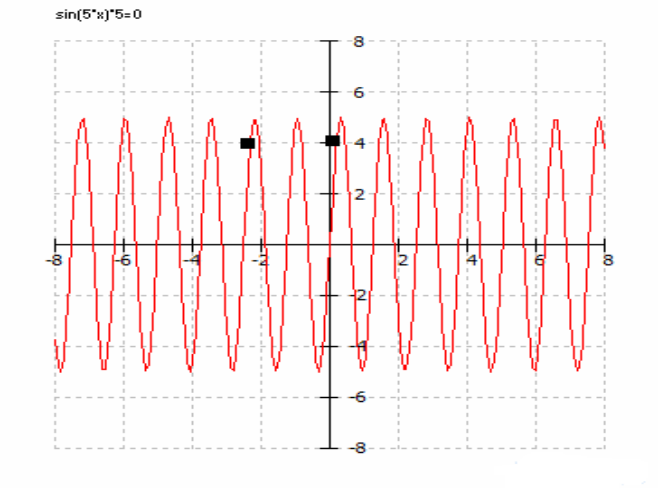
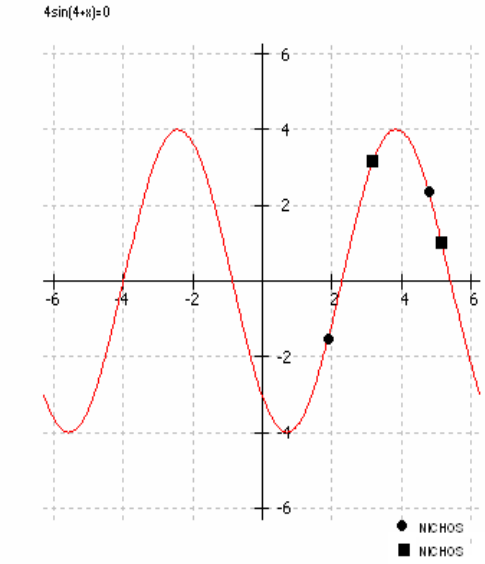


Fig 6.69 Está muy cerca los máximo

F Más de un nicho buscas el mismo máximo Si varios nichos están buscado el mismo máximo y están bastante próximos, la librería lo que realiza es quedarse con un sólo nicho, eligiendo aquellos puntos con mayor fitness y eliminando el resto de nichos que busca el mismo máximo. Se puede observar en la figura 6.70 como dos nichos están buscando el mismo máximo y como en la figura 6.71 se elimina uno de los nichos y se modifica con los mejores fitness.



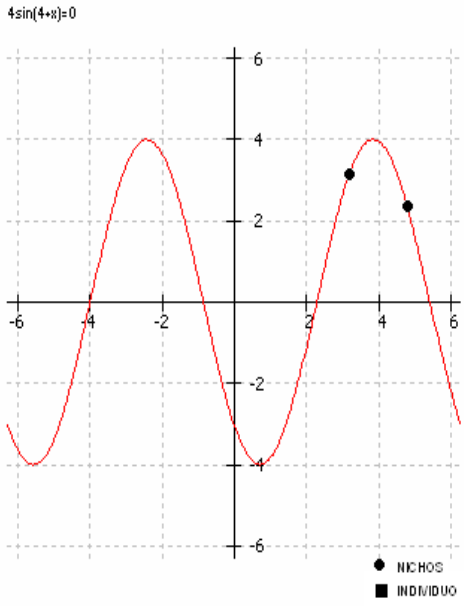


Fig 6.70 Dos nichos buscan la misma solución **Fig 6.71 Elimina un nicho y se queda con los mejores valores**

El mismo caso de uso se puede aplicar en la figura 6.72 en la cual dos nichos buscan el mismo máximo y cómo se elimina un nicho, quedándose con los mejores valores de los dos nichos en la figura 6.73.

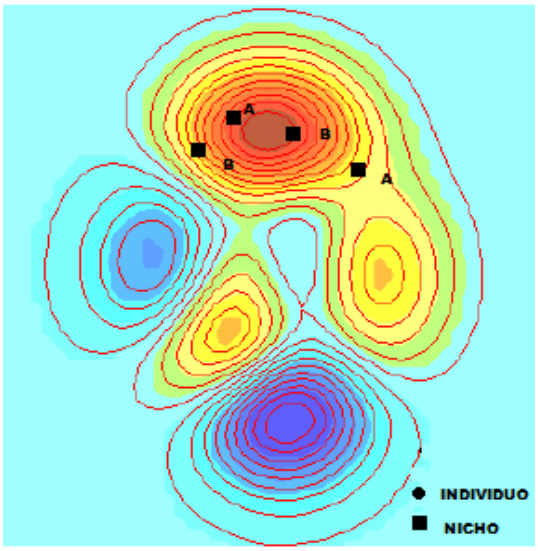


Fig 6.72 Dos nichos buscan la misma solución

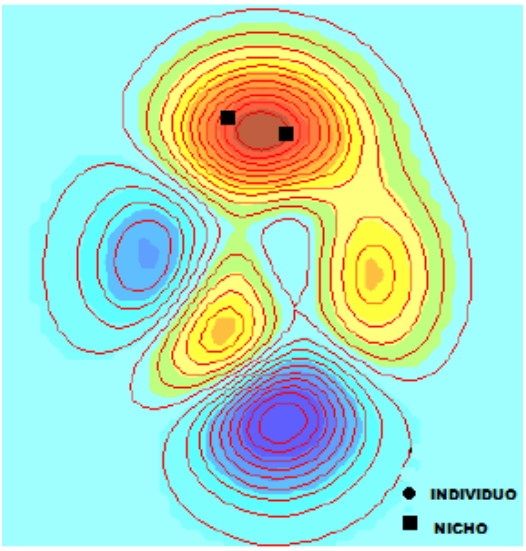


Fig 6.73 Elimina un nicho y con los mejores valores

G Los puntos del nicho están alejados. Si ya se ha ejecutado varias generaciones y existe algún nicho con los puntos que lo forman y están alejados. La librería modificará o eliminaría el nicho dependiendo de los valores almacenados del nicho. Para saber si un nicho tienes sus valores próximos o lejanos se utilizará la constante de radio nicho que se ha hablado anteriormente Ahora se indica todos los posibles casos de usos que se pueden dar:

G001 Los puntos del nicho tienen un fitness elevado. En este caso de uso los puntos que forman el nicho están bastante alejados y tienen ambos un valor de fitness alto. Seguramente puede ocurrir este caso porque entre los valores del nicho se encuentre más de un máximo. Entonces se dividirá el nicho en dos, uno de los nichos contendrá el valor del nicho original y otro un valor aleatorio próximo a ese punto. Y el otro nicho contendrá el otro punto con otro valor aleatorio próximo a él. En la figura 6.74 se puede observar que el nicho está formado por puntos con un alto fitness y está bastante alejado. Y cómo en la figura 6.75 divide ese nicho en dos nuevos nichos.

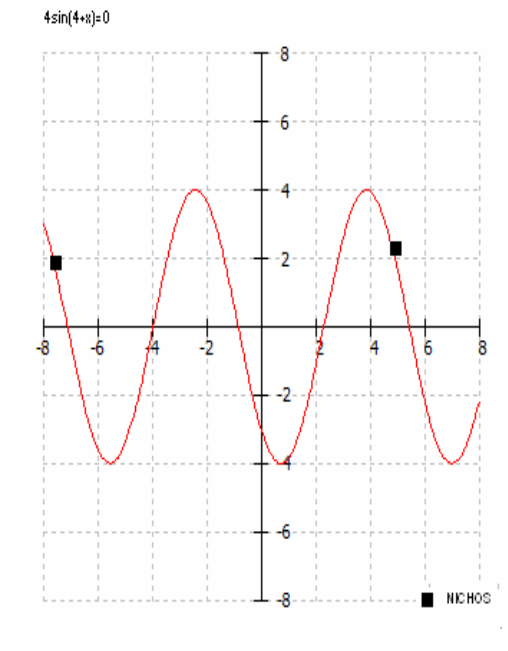


Fig 6.74 Un nichos alto fitness y muy distanciados

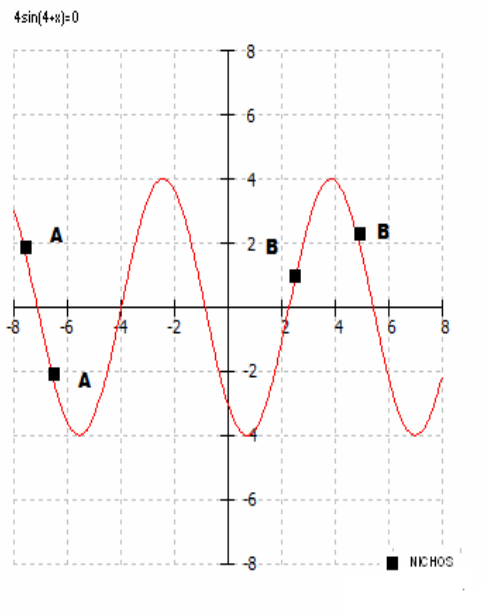


Fig6.75 Se divide el nicho creándose dos nichos

Es el mismo caso anterior, como se puede observar en la figura 6.76 los puntos que forman el nicho se encuentra a una cierta distancia y poseen los dos un fitness elevado, entonces se divide el nicho en dos nuevos como se muestra en la figura 6.77

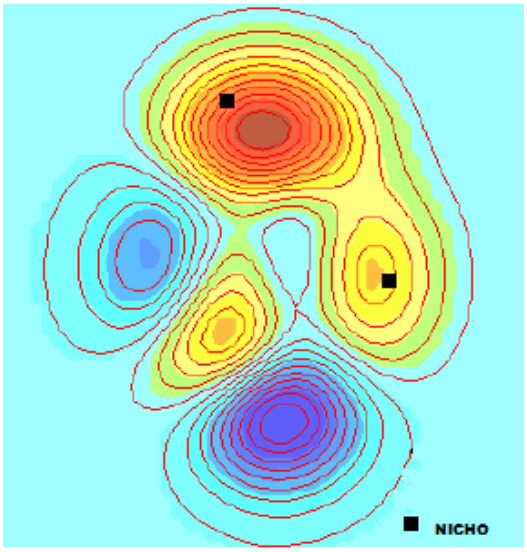


Fig 6.76 Un nichos alto fitness y muy distanciados

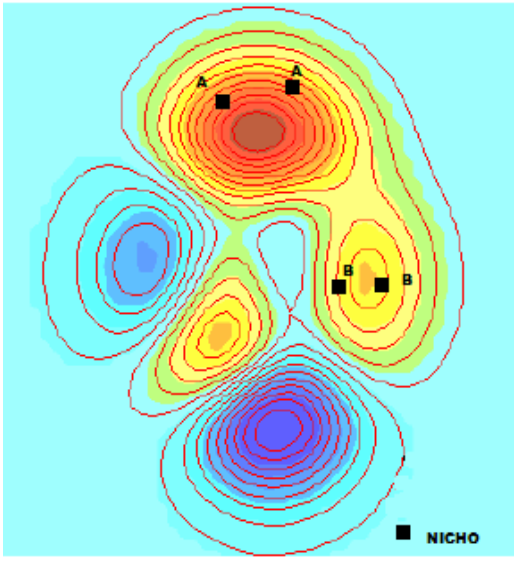


Fig6.77 Se divide el nicho creándose dos nichos

G002 Unos de puntos del nicho tiene un fitness elevado y el otro un fitness bajo. Después de ejecutarse varias generaciones si existe algún nicho que sus puntos estén alejados y unos de su puntos tenga un valor alto fitness y el otro no, el algoritmo intentará eliminar el punto que tiene menos fitness por un valor aleatorio con igual o mayor valor de fitness pero que esté más cerca que el punto con mayor fitness. Se puede observar en la figura 6.78 como el nicho posee un punto fitness elevado y otro no. Se elimina el punto con menor fitness y lo sustituye por un punto más cercano como se puede observar en la figura 6.79.

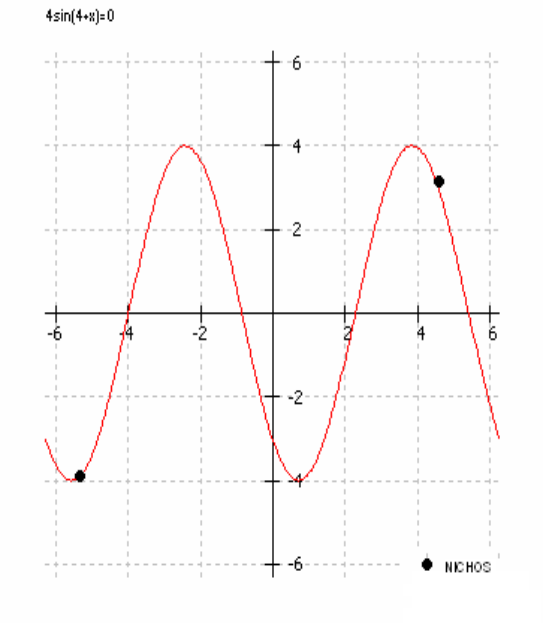


Fig 6.78 Unos nichos distanciados y fitness diferente

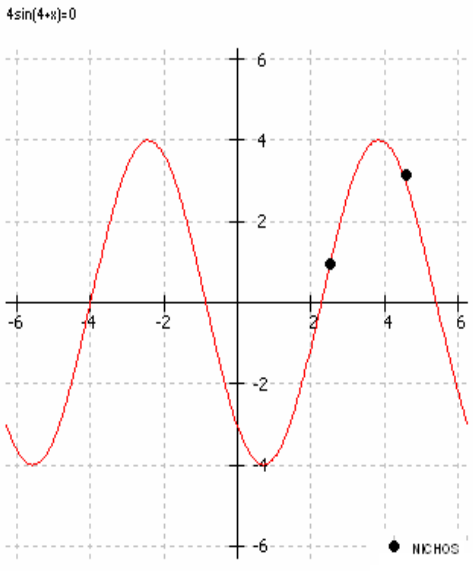


Fig 6.79 Se modifica el nicho con un valor aleatorio

Se puede advertir este mismo caso uso pero esta vez en la figura 6.80 y cómo es sustituido el punto con menor fitness por otro punto más cercano al individuo (figura 6.81)

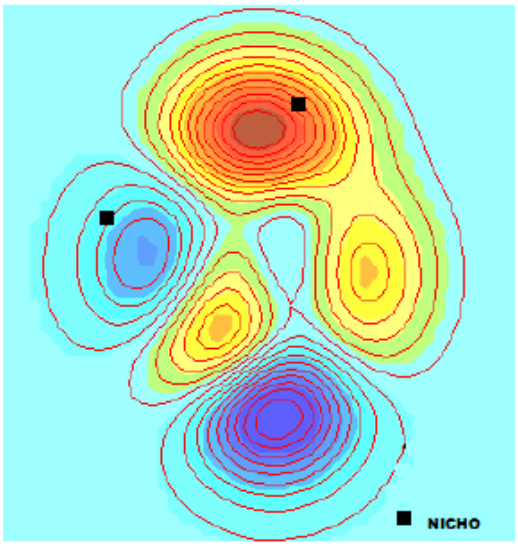


Fig 6.80 Unos nichos distanciados y fitness diferente

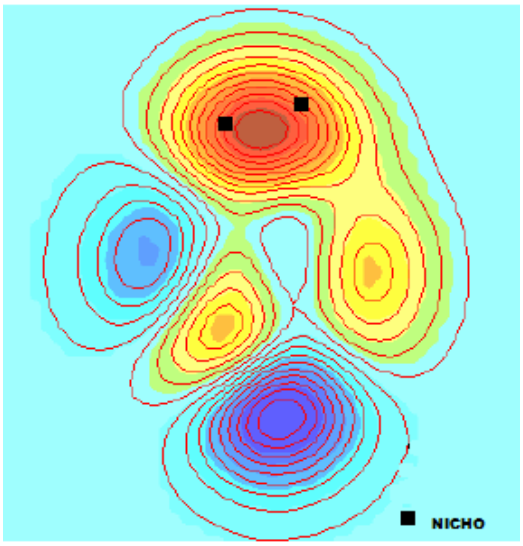


Fig 6.81 Se modifica el nicho con un valor aleatorio

G003 Los puntos que forma el nicho tiene un fitness bajo. Si los puntos que forman un nicho están alejados y además poseen un fitness bajo se puede eliminar el nicho a no está cerca de ninguna solución como se puede observar en la figura 6.82 y 6.83.

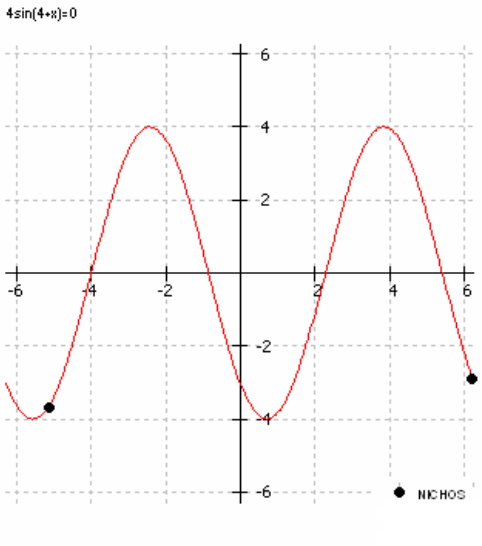


Fig 6.82 Uno nicho con fitness bajos y distanciados

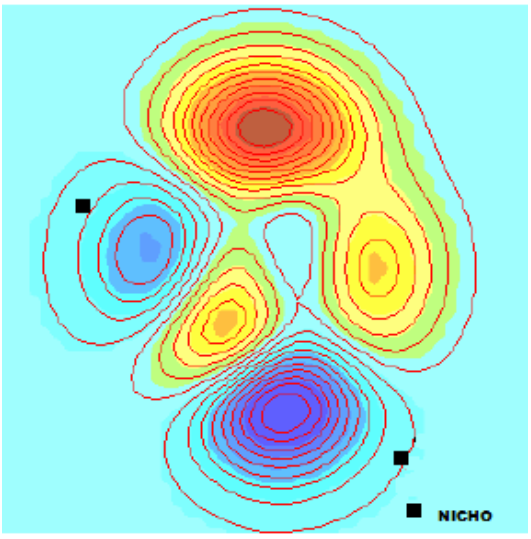


Fig 6.83 Nicho con los puntos distanciados y fitness bajo

6.2 Pruebas con la librería desarrollada

En este capítulo se presentarán los resultados experimentales del estudio realizado para comprobar cómo el algoritmo obtiene los máximos de ciertas ecuaciones. Para realizar dicho estudio se obtienen los valores de la población en diferentes tramos de la ejecución del algoritmo, concretamente en la generación 30, en la generación 60 y en la generación 90 (al finalizar el algoritmo). Se han seleccionado estos tramos para analizar, la evolución de la población a lo largo de la evolución: el inicio, la mitad y el final del algoritmo. De esta forma se puede comparar y ver cómo se modifica la población según se va ejecutando el algoritmo.

El estudio del algoritmo se llevará a cabo con las siguientes funciones:

- Función x.1: Una función sencilla, con una variable de entrada. Si el rango de la función es [-8,8] existe dos máximos en los valores x=-2.429 y x=3.853

$$Y = 4 * sen(4 + x)$$

- Función x.2: Una función más complicada, con dos variables de entrada. Aquí nos encontramos 3 máximos en los valores (-0.460,-0.629), (1.286,-0.005) y (-0.009,1.581)

$$f(x,y) = 3(1-x)^2 e^{-(x^2+(y+1)^2)} - 10(\frac{x}{5} - x^3 - y^5)e^{-(x^2+y^2)} - \frac{1}{3}e^{-((x+1)^2+y^2)}$$

6.2.1 Resultados experimentales con la Función x.1

En la tabla 6.1 y en la figura 6.84 se muestran los valores de la población en la generación 30. En el figura 6.84, el eje x está representado el individuo. Los individuos van desde el individuo 0 al individuo 99. En el eje y, se indican los diferentes valores de la función para cada individuo. Se intenta buscar los valores x, donde se convierte la función en máximos.
Se puede observar que población está muy dispersa, no dando ningún indicio con qué valores de x podrían encontrarse los máximos. Si realizamos el cálculo de la desviación típica de la población, en la generación 30 se obtiene un valor de 4.515, siendo bastante elevada.

La razón por la que la población está tan dispersa es porque al inicio, el algoritmo intenta crear el máximo número de nichos para localizar todos los posibles máximos. Cuanto más expandida esté la población más fácil es encontrar todos las posibles soluciones.

GENERACIÓN 30							
NUM	VALOR X	NUM	VALOR X	NUM	VALOR X	NUM	VALOR X
0	3,885	25	2,960	50	-4,214	75	-4,804
1	3,899	26	-3,327	51	-6,921	76	-6,331
2	3,805	27	-7,776	52	-4,260	77	-4,834
3	-2,353	28	-7,764	53	-6,867	78	6,260
4	-2,518	29	-7,721	54	2,007	79	6,329
5	3,758	30	2,858	55	1,984	80	0,087
6	-2,308	31	4,860	56	-0,484	81	0,120
7	3,704	32	2,799	57	-6,751	82	1,232
8	4,013	33	2,708	58	-6,743	83	-6,081
9	-2,590	34	5,048	59	1,859	84	0,206
10	3,661	35	-1,198	60	1,851	85	1,203
11	-2,637	36	-3,681	61	1,811	86	7,427
12	-2,699	37	2,602	62	8,056	87	-5,930
13	4,155	38	2,599	63	-0,341	88	7,305
14	-2,840	39	2,575	64	-0,268	89	7,266
15	3,350	40	-1,075	65	-0,232	90	7,260
16	4,368	41	5,240	66	-4,641	91	0,450
17	4,371	42	2,452	67	-6,445	92	0,960
18	4,392	43	-3,883	68	1,579	93	0,947
19	-3,000	44	-3,916	69	1,526	94	0,917
20	-8,101	45	-3,960	70	6,186	95	0,904
21	-1,749	46	5,440	71	-0,091	96	0,894
22	4,546	47	-4,103	72	6,204	97	6,867
23	-1,696	48	-0,715	73	7,763	98	-5,512
24	-7,908	49	-6,949	74	7,763	99	7,023

Tabla 6.1 Datos de la generación 30

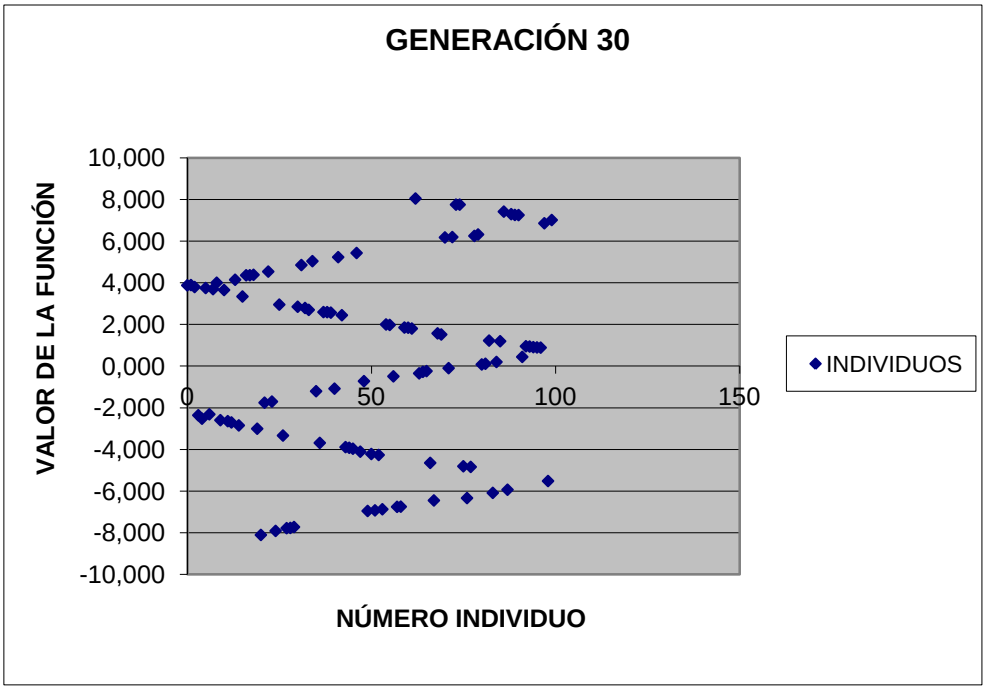


Fig 6.84 Gráfica de la generación 30

El siguiente paso será estudiar la población cuando ya se ha ejecutado más de la mitad del algoritmo, es decir realizar el estudio en la generación 60. En esta generación se puede comprobar que existen dos nichos, y la población se ha repartido entre esas dos sub-poblaciones que se han creado entre los nichos. De esta forma, la población se va acotando para encontrar las posibles soluciones al problema.

Si se observa la tabla 6.2 se puede comprobar que los valores de x se van agrupando alrededor de dos valores.

GENERACIÓN 60							
NUM	VALOR X	NUM	VALOR X	NUM	VALOR X	NUM	VALOR X
0	3,854	25	3,853	50	3,854	75	3,853
1	3,853	26	3,853	51	3,853	76	-2,429
2	-2,429	27	-2,433	52	-2,431	77	-2,432
3	-2,429	28	-2,431	53	-2,431	78	3,854
4	3,854	29	3,854	54	3,854	79	-2,431
5	-2,432	30	3,853	55	3,853	80	-2,432
6	-2,430	31	-2,433	56	-2,435	81	3,853
7	3,853	32	-2,432	57	-2,430	82	-2,432
8	-2,431	33	3,853	58	-2,431	83	3,854
9	3,853	34	-2,430	59	3,854	84	3,854
10	3,854	35	3,853	60	-2,435	85	-2,435
11	3,854	36	3,854	61	-2,435	86	3,854
12	-2,431	37	3,854	62	-2,432	87	3,853
13	3,853	38	3,854	63	-2,429	88	3,853
14	-2,429	39	-2,436	64	3,853	89	-2,434
15	-2,433	40	-2,436	65	3,854	90	-2,433
16	-2,435	41	3,853	66	-2,434	91	3,854
17	-2,433	42	-2,436	67	-2,433	92	3,853
18	-2,434	43	3,853	68	3,854	93	3,853
19	-2,436	44	-2,431	69	-2,433	94	3,853
20	-2,434	45	3,853	70	3,854	95	3,854
21	-2,433	46	-2,432	71	3,853	96	3,853
22	-2,433	47	3,853	72	-2,434	97	3,853
23	3,854	48	3,854	73	-2,432	98	3,854
24	-2,433	49	-2,435	74	-2,429	99	3,853

Tabla 6.2 Datos de la generación de 60

En la figura 6.85 se puede ver que los valores de x se han ido agrupando a dos valores. En este caso para calcular la desviación típica se divide la población en dos sub-poblaciones, una que se acerca a los valores –3.8 y la otra a los valores que se acercan a –2.4. Realizando los cálculos de los primeros individuos existe una desviación típica de 0.0020. En cambio, para la segunda sub-población la desviación típica es de 0.0004

Según la figura 6.85 y los datos que se reflejan en la tabla 6.2, la población se va acercando a la solución del problema.

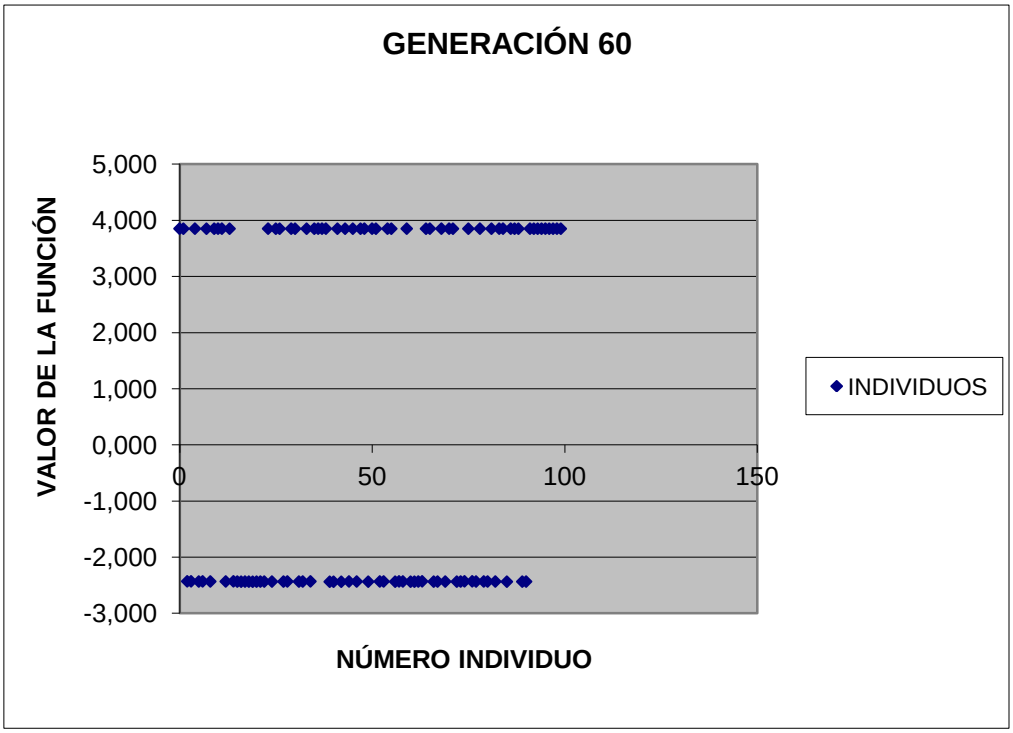


Fig 6.85 Gráfica de la generación 60

A continuación se realizará el estudio en la generación 90, generación en la que casi ha finalizado el algoritmo. La idea es comprobar si los individuos de la población siguen acercándose a los valores obtenidos en la generación 60 o por el contrario han aparecido otros valores. Como se muestra en la tabla 6.3, los individuos de la población se van agrupando en torno a los valores a 3.8 y –2.4.

GENERACIÓN 90							
NUM	VALOR X	NUM	VALOR X	NUM	VALOR X	NUM	VALOR X
0	-2,431	25	3,853	50	-2,432	75	3,853
1	-2,431	26	-2,429	51	3,853	76	3,853
2	3,853	27	-2,429	52	3,853	77	-2,432
3	-2,431	28	3,853	53	-2,431	78	3,853
4	-2,430	29	-2,430	54	3,853	79	3,853
5	3,853	30	-2,430	55	3,853	80	-2,429
6	3,853	31	-2,432	56	-2,429	81	3,853
7	3,853	32	-2,429	57	-2,430	82	-2,429
8	3,853	33	-2,431	58	3,853	83	3,853
9	-2,432	34	-2,432	59	3,853	84	-2,429
10	3,853	35	-2,430	60	-2,430	85	-2,431
11	3,853	36	3,853	61	3,853	86	-2,432
12	3,853	37	3,853	62	-2,430	87	3,853
13	3,853	38	-2,431	63	3,853	88	-2,429
14	3,853	39	3,853	64	3,853	89	3,853
15	3,853	40	3,853	65	-2,431	90	-2,429
16	3,853	41	3,853	66	3,853	91	-2,429
17	3,853	42	-2,432	67	-2,430	92	-2,432
18	3,853	43	3,853	68	-2,429	93	3,853
19	3,853	44	3,853	69	3,853	94	-2,431
20	-2,431	45	-2,432	70	3,853	95	-2,432
21	3,853	46	-2,432	71	-2,432	96	-2,429
22	-2,432	47	-2,431	72	3,853	97	-2,431
23	-2,429	48	3,853	73	-2,432	98	3,853
24	3,853	49	-2,432	74	-2,431	99	-2,432

Tabla 6.3 Datos de la generación de 90

Como se muestra en la figura 6.86 los individuos de la población se van agrupando en dos valores, dividiéndose la población en dos sub-poblaciones, una población formada por valores cercanos al 3.8 y la otra población formada los individuos cercanos a –2.4. Además, estas sub-poblaciones están formadas por 50 individuos cada una. Si se calcula la desviación típica de la primera sub-población, aproximadamente es cero y de la segunda sub-población su desviación típica es 0.0011.

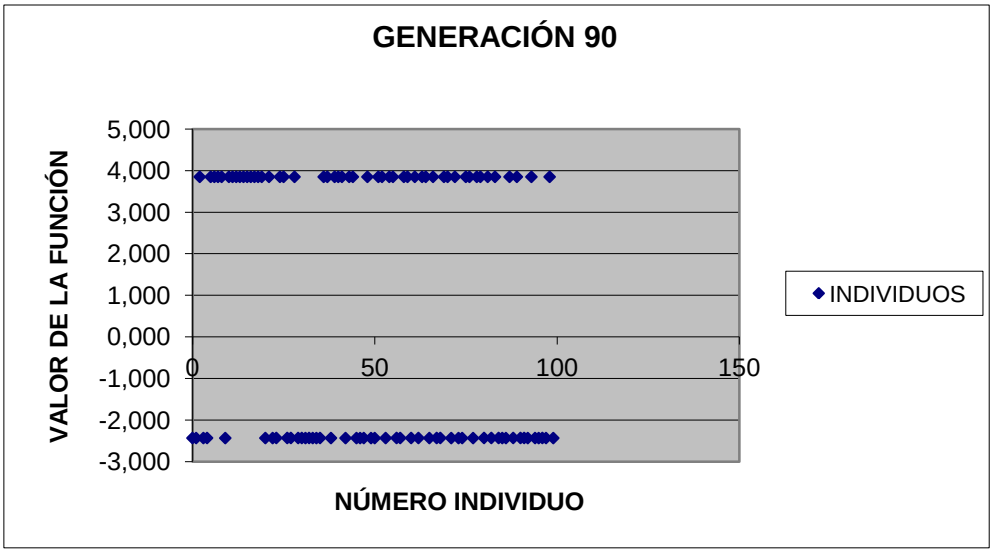


Fig 6.86 Gráfica de la generación 90

Para terminar el estudio con la primera función, en la tabla 6.4 se muestran los valores obtenidos una vez finalizada el algoritmo. Comparado los datos de la generación 90 se puede comprobar que no se han obtenido mejores resultados.

GENERACIÓN FINAL							
NUM	VALOR X	NUM	VALOR X	NUM	VALOR X	NUM	VALOR X
0	-2,432	25	-2,430	50	-2,430	75	3,853
1	-2,431	26	-2,432	51	-2,431	76	3,853
2	3,853	27	-2,432	52	-2,432	77	3,853
3	3,853	28	-2,431	53	-2,430	78	3,853
4	-2,432	29	-2,429	54	-2,431	79	3,853
5	3,853	30	3,853	55	-2,431	80	3,853
6	-2,432	31	-2,429	56	-2,429	81	3,853
7	3,853	32	-2,431	57	-2,430	82	3,853
8	-2,430	33	3,853	58	3,853	83	3,853
9	-2,432	34	-2,432	59	3,853	84	3,853
10	-2,432	35	-2,430	60	3,853	85	3,853
11	-2,432	36	3,853	61	3,853	86	3,853
12	3,853	37	-2,430	62	3,853	87	3,853
13	-2,429	38	-2,432	63	-2,429	88	3,853
14	-2,432	39	-2,431	64	-2,432	89	3,853
15	-2,429	40	3,853	65	-2,429	90	3,853
16	-2,431	41	-2,432	66	3,853	91	3,853
17	-2,432	42	3,853	67	-2,429	92	3,853
18	-2,429	43	-2,430	68	3,853	93	3,853
19	-2,430	44	-2,430	69	-2,429	94	3,853
20	-2,431	45	-2,429	70	3,853	95	3,853
21	-2,429	46	3,853	71	3,853	96	3,853
22	-2,432	47	3,853	72	3,853	97	3,853
23	-2,431	48	-2,432	73	3,853	98	3,853
24	-2,431	49	3,853	74	3,853	99	3,853

Tabla 6.4 Datos de la generación final

Como muestra la figura 6.87, la población se ha agrupado en dos valores, –2.4 y 3.8. Estos dos valores son soluciones aproximadas al problema de buscar los máximos de la función x.1. Los máximos de la función x.1 se encuentran en -2,429 y en 3,853. Los resultados obtenidos por la librería son bastante próximos

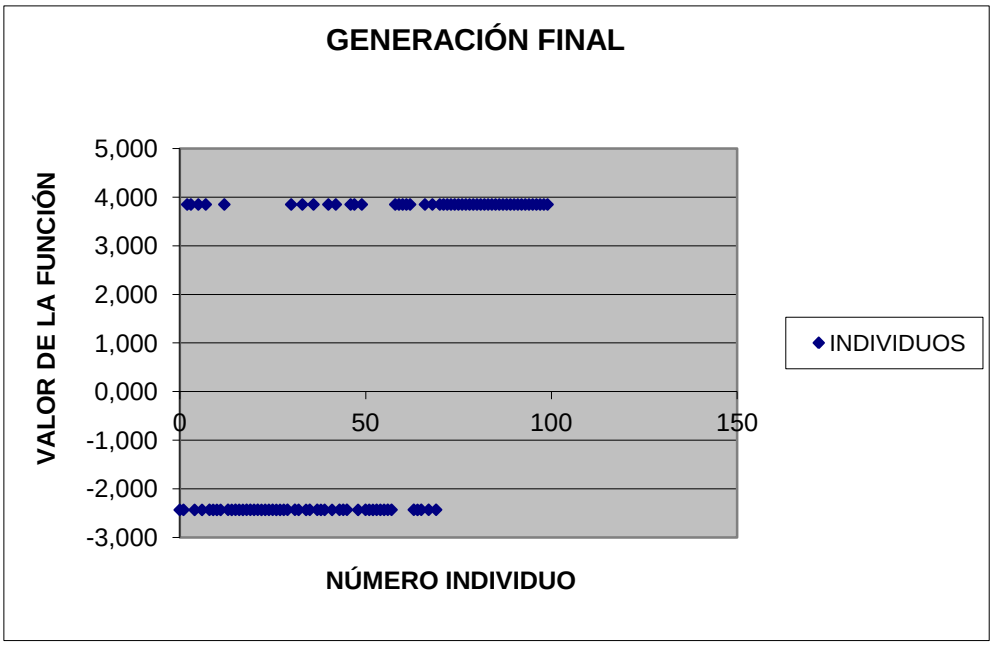


Fig 6.87 Gráfica de la generación final

6.2.2 Resultados experimentales con la Función x.2

Después de realizar el estudio con una función sencilla de una variable, ahora se realizará el estudio con un problema de mayor dificultad. Se utiliza la función dada por la función x.2 para comprobar y ver cómo el algoritmo resuelve este tipo de problemas y cómo va evolucionando la población hasta obtener las soluciones de los máximos de esta función. Para obtener los máximos de esta función se debería realizar diferentes derivadas. Además son derivadas complicadas. Por esta razón, se llevará a cabo la resolución de este problema con los algoritmos genéticos con nichos.

Como ocurría en la ejecución anterior, en las primeras generaciones la población está muy expandida como muestra la figura 6.88 y la tabla 6.5. Al inicio, el algoritmo trata de expandir lo máximo posible la población para la creación de los diferentes nichos, y obtener todas las posibles soluciones al problema.

GENERACIÓN 30					
NUM	VALOR X	VALOR Y	NUM	VALOR X	VALOR Y
0	0,475	1,826	50	6,851	-7,548
1	1,249	0,066	51	-6,051	7,780
2	-0,874	2,042	52	7,997	5,505
3	-0,093	0,774	53	6,046	-8,107
4	-0,859	2,881	54	-7,924	-5,594
5	2,097	2,338	55	4,951	7,956
6	-1,808	-2,214	56	5,737	-7,547
7	-1,586	3,283	57	6,215	6,673
8	-1,571	-2,880	58	-6,928	-6,122
9	0,083	3,992	59	6,522	5,942
10	2,717	3,058	60	5,634	-7,194
11	-3,042	3,437	61	7,961	-3,661
12	4,267	1,500	62	-3,471	8,008
13	-1,192	-5,289	63	-4,251	-7,845
14	-1,178	5,150	64	-5,491	-6,740
15	3,566	-3,663	65	4,289	7,210
16	-0,890	5,288	66	6,078	-5,948
17	-2,727	-4,884	67	-3,801	7,301
18	-1,115	5,406	68	5,912	-5,960
19	4,955	1,982	69	-0,213	8,185
20	-4,496	3,147	70	7,644	-1,860
21	5,312	1,784	71	-2,239	7,777
22	5,571	-0,862	72	-3,620	7,192
23	4,913	3,013	73	4,991	-6,397
24	-4,301	-4,626	74	-5,669	-5,666
25	-3,742	-5,209	75	5,008	-6,253
26	4,585	4,024	76	-5,098	5,719
27	-1,984	-6,242	77	0,082	7,738
28	2,189	5,880	78	7,128	1,804
29	1,798	-5,939	79	0,470	7,598
30	5,811	1,816	80	6,614	3,148
31	-1,134	-6,725	81	-5,650	4,139
32	4,685	-4,680	82	0,388	7,419
33	4,605	4,582	83	-5,562	-4,758
34	5,763	-3,152	84	6,751	0,845
35	-6,393	2,846	85	6,781	-0,354
36	-6,821	2,571	86	4,117	-5,932
37	-7,380	-1,625	87	3,062	-6,477
38	-7,555	-1,766	88	-2,365	6,545
39	-7,033	-3,547	89	-6,629	-1,454
40	-7,900	-2,795	90	-6,352	-1,548
41	-8,015	3,431	91	0,559	-6,385
42	-6,558	5,486	92	-6,391	-0,465
43	-7,140	5,438	93	-6,278	1,185
44	-6,999	6,204	94	-6,296	-1,070
45	7,839	7,188	95	-5,344	1,335
46	7,957	6,765	96	-4,214	-0,374
47	-6,904	-8,029	97	1,835	-3,670
48	6,851	-7,548	98	2,243	-2,828
49	-6,051	7,780	99	-3,141	0,910

Tabla 6.5 Datos de la generación 30 de la función x.2.

Con los valores de la tabla 6.5, se ha calculado la desviación típica de los valores de x cuyo resultado es 5.0001 y la desviación típica de los valores de y es 4.9867.

En la figura 6.88 se representan los individuos en la generación 30, cada punto representando la posición del individuo. Se observa que en esta generación los individuos no se están agrupados en ninguna zona del espacio de entrada, estos dispersos por todo el espacio. Esto es conveniente para localizar y la crear de nichos. Cuantos más dispersa está la población a inicio del algoritmo, más posibilidades se tiene de encontrar todos los posibles máximos.

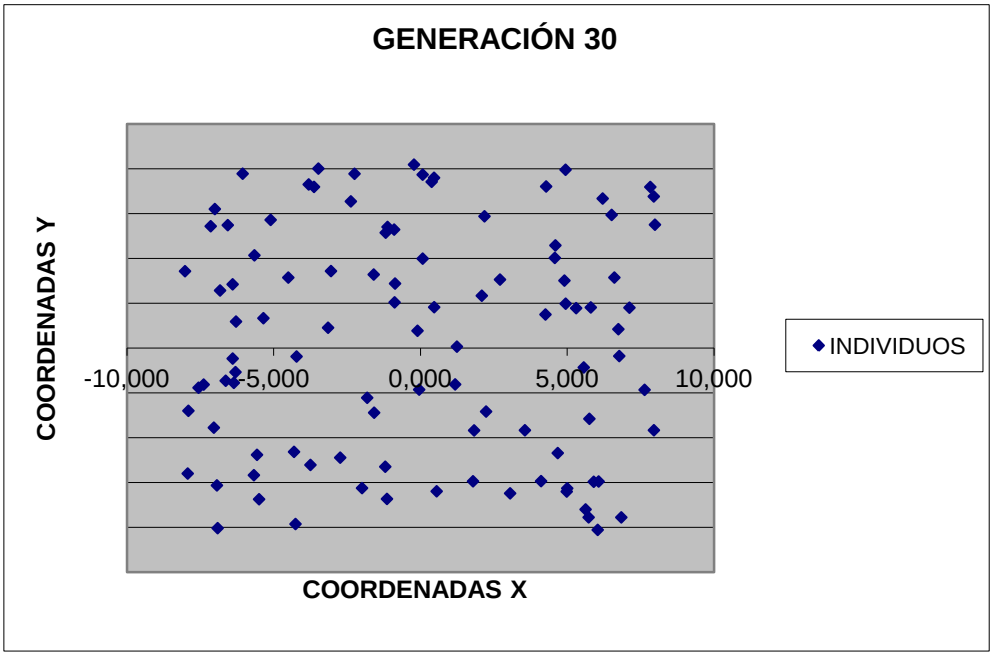


Fig 6.88 Localización de los individuos de la población en la generación 30

La siguiente toma de datos se realiza en la generación 60, en la que el algoritmo ya se ejecutado más de la mitad de las generaciones. En esta generación los individuos de la población ya han empezado agrupase en algunos valores como se puede comprobar en la tabla 6.6 Se observa que se han agrupado en tres zonas diferentes del espacio del entrada. Esto se puede observar también en la figura 6.89.

GENERACIÓN 60					
NUM	VALOR X	VALOR Y	NUM	VALOR X	VALOR Y
0	-0,068	1,487	50	-0,116	1,428
1	1,250	0,134	51	-0,183	1,479
2	1,265	0,081	52	-0,373	-0,774
3	-0,087	1,498	53	0,075	1,427
4	-0,485	-0,776	54	-0,101	1,430
5	-0,143	1,458	55	-0,130	1,420
6	-0,101	1,416	56	-0,076	1,409
7	0,015	1,498	57	0,063	1,410
8	-0,060	1,443	58	0,063	1,460
9	-0,135	1,454	59	1,256	0,152
10	-0,167	1,434	60	-0,418	-0,777
11	1,282	0,114	61	-0,009	1,446
12	1,297	0,140	62	-0,213	1,457
13	1,290	0,106	63	-0,162	1,499
14	-0,120	1,494	64	1,251	0,129
15	-0,011	1,422	65	0,091	1,437
16	1,297	0,082	66	-0,107	1,416
17	1,277	0,139	67	-0,069	1,428
18	1,273	0,104	68	-0,135	1,423
19	-0,184	1,438	69	1,255	0,104
20	-0,483	-0,783	70	-0,514	-0,813
21	0,014	1,410	71	1,279	0,104
22	-0,416	-0,772	72	-0,482	-0,805
23	1,286	0,121	73	1,298	0,121
24	-0,192	1,409	74	-0,419	-0,828
25	1,261	0,073	75	1,282	0,149
26	0,075	1,433	76	-0,472	-0,825
27	1,276	0,147	77	-0,498	-0,855
28	1,265	0,146	78	1,267	0,136
29	-0,404	-0,805	79	-0,489	-0,813
30	1,264	0,138	80	1,259	0,069
31	1,261	0,122	81	1,282	0,098
32	-0,348	-0,797	82	-0,368	-0,795
33	1,279	0,110	83	1,270	0,111
34	-0,202	1,423	84	1,299	0,081
35	-0,388	-0,845	85	-0,340	-0,832
36	1,254	0,135	86	-0,388	-0,777
37	-0,518	-0,775	87	-0,499	-0,776
38	-0,398	-0,855	88	1,273	0,153
39	1,257	0,147	89	-0,382	-0,832
40	1,280	0,128	90	-0,474	-0,821
41	1,268	0,104	91	-0,505	-0,798
42	-0,523	-0,840	92	-0,482	-0,814
43	1,300	0,120	93	-0,450	-0,815
44	1,274	0,143	94	-0,367	-0,789
45	0,059	1,407	95	-0,347	-0,850
46	-0,388	-0,806	96	-0,506	-0,796
47	-0,044	1,427	97	-0,498	-0,773
48	-0,221	1,430	98	-0,509	-0,808
49	-0,399	-0,786	99	-0,118	1,484

Tabla 6.6 Datos de la generación 60 de la función x.2.

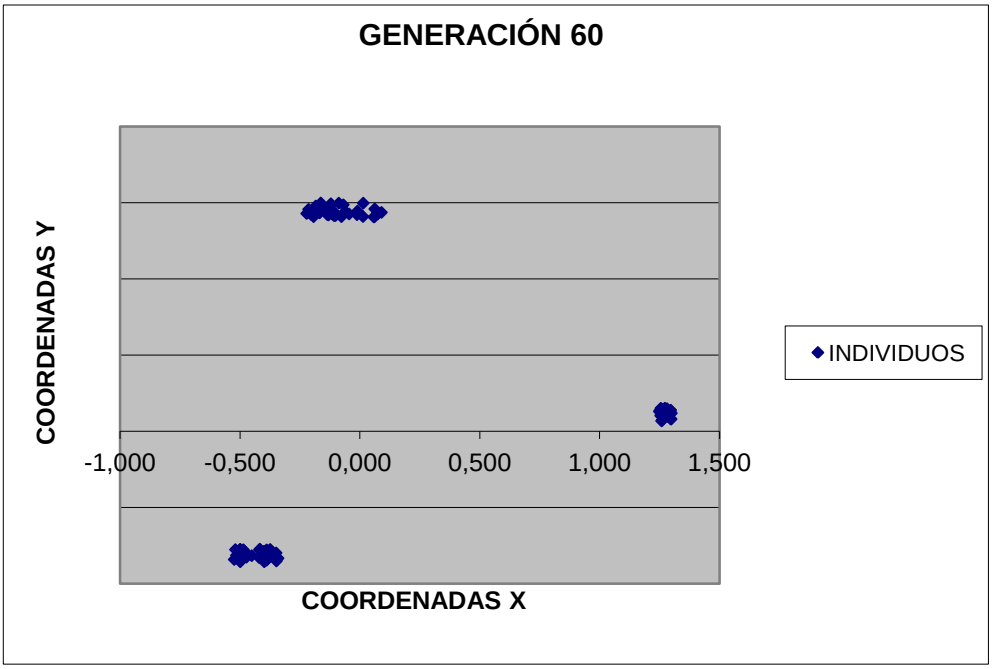


Fig 6.89 Localización de los individuos de la población en la generación 60

Al calcular la desviación típica con los resultados de la tabla 6.7 se obtiene un valor de 0.737 para la coordenada x y un valor de 0.926 para la coordenada y. Pero realizando un estudio más en profundidad y dividiendo la población en tres sub-poblaciones, la desviación típica de estas sub-poblaciones se reduce bastante como se puede observar en la tabla 6.6.

	POBLACIÓN 1	POBLACIÓN 2	POBLACIÓN 3
Desviación típica X	0.058752942	0.091775306	0.014591593
Desviación típica Y	0.025453499	0.028734557	0.023827703

Tabla 6.7. Desviación típica de las sub-poblaciones en la generación 60

Como para la función x.1, también se ha recogidos los resultados de la generación 90, en la que se esté finalizado la ejecución del algoritmo. A primera vista, si se comparan los resultados de la tabla 6.8 con los resultados en la tabla 6.6 (valores para la generación 60) no se han obtenido grandes cambios.

GENERACIÓN 90					
NUM	VALOR X	VALOR Y	NUM	VALOR X	VALOR Y
0	-0,024	1,595	50	-0,478	-0,583
1	1,218	0,053	51	1,244	0,062
2	1,245	0,058	52	1,232	0,055
3	-0,005	1,509	53	-0,109	1,614
4	-0,464	-0,568	54	1,227	0,064
5	-0,416	-0,580	55	-0,485	-0,575
6	0,011	1,630	56	1,233	0,054
7	0,050	1,562	57	1,218	0,063
8	-0,014	1,652	58	0,005	1,650
9	1,242	0,057	59	1,234	0,061
10	-0,002	1,580	60	0,001	1,646
11	-0,051	1,678	61	1,245	0,056
12	-0,359	-0,566	62	-0,392	-0,564
13	1,246	0,057	63	-0,386	-0,568
14	0,058	1,559	64	0,016	1,602
15	-0,461	-0,575	65	1,247	0,061
16	1,240	0,064	66	-0,089	1,547
17	1,245	0,062	67	-0,384	-0,565
18	1,239	0,063	68	-0,007	1,643
19	1,242	0,056	69	0,047	1,573
20	-0,386	-0,583	70	0,027	1,609
21	-0,038	1,568	71	1,228	0,059
22	1,226	0,057	72	1,239	0,057
23	0,030	1,671	73	-0,456	-0,587
24	-0,066	1,648	74	1,222	0,061
25	-0,049	1,520	75	-0,087	1,515
26	1,226	0,061	76	1,223	0,054
27	-0,365	-0,565	77	-0,426	-0,564
28	-0,475	-0,564	78	1,243	0,060
29	1,219	0,053	79	1,231	0,057
30	-0,067	1,604	80	-0,031	1,508
31	1,235	0,053	81	-0,027	1,544
32	-0,360	-0,564	82	-0,091	1,623
33	1,229	0,062	83	-0,345	-0,572
34	-0,438	-0,569	84	-0,408	-0,592
35	-0,365	-0,566	85	-0,342	-0,586
36	1,231	0,055	86	-0,430	-0,587
37	-0,341	-0,569	87	-0,462	-0,586
38	-0,008	1,604	88	-0,428	-0,562
39	-0,069	1,504	89	-0,328	-0,577
40	1,235	0,062	90	-0,438	-0,580
41	-0,395	-0,569	91	-0,353	-0,579
42	-0,013	1,528	92	-0,403	-0,587
43	0,027	1,565	93	-0,385	-0,573
44	-0,021	1,536	94	-0,425	-0,572
45	-0,103	1,556	95	-0,370	-0,566
46	0,024	1,625	96	1,226	0,053
47	0,040	1,548	97	1,222	0,059
48	-0,331	-0,571	98	1,220	0,060
49	1,239	0,057	99	0,090	1,552

Tabla 6.8 Datos de la generación 90 de la función x.2.

Con los datos obtenidos en la tabla 6.9 se calcula la desviación típica total de la variable x y de la variable y, obteniendo 0.69615 y 0.91000, respectivamente. Es una desviación bastante alta. Pero como en el caso anterior, si se divide la población en tres sub-poblaciones y se vuelve a realizar el estudio de las desviaciones típicas, se obtienen

los datos de la tabla 6.9, en la que puede comprobar que la desviación típica se reduce bastante, agrupándose la población en tres puntos.

	POBLACIÓN 1	POBLACIÓN 2	POBLACIÓN 3
Desviación típica X	0.045844289	0.049080058	0.009105241
Desviación típica Y	0.008616511	0.049977573	0.003444963

Tabla 6.9. Desviación típica de las sub-poblaciones de la generación 90

Comparando las tablas de desviación típica 6.7 y 6.9 se puede comprobar que en esta última generación se han agrupado aún más los individuos de las sub-poblaciones.

Como se puede observar en la figura 6.90, la población se va concentrando en tres zonas o puntos, que son la solución al problema que se han planteado.

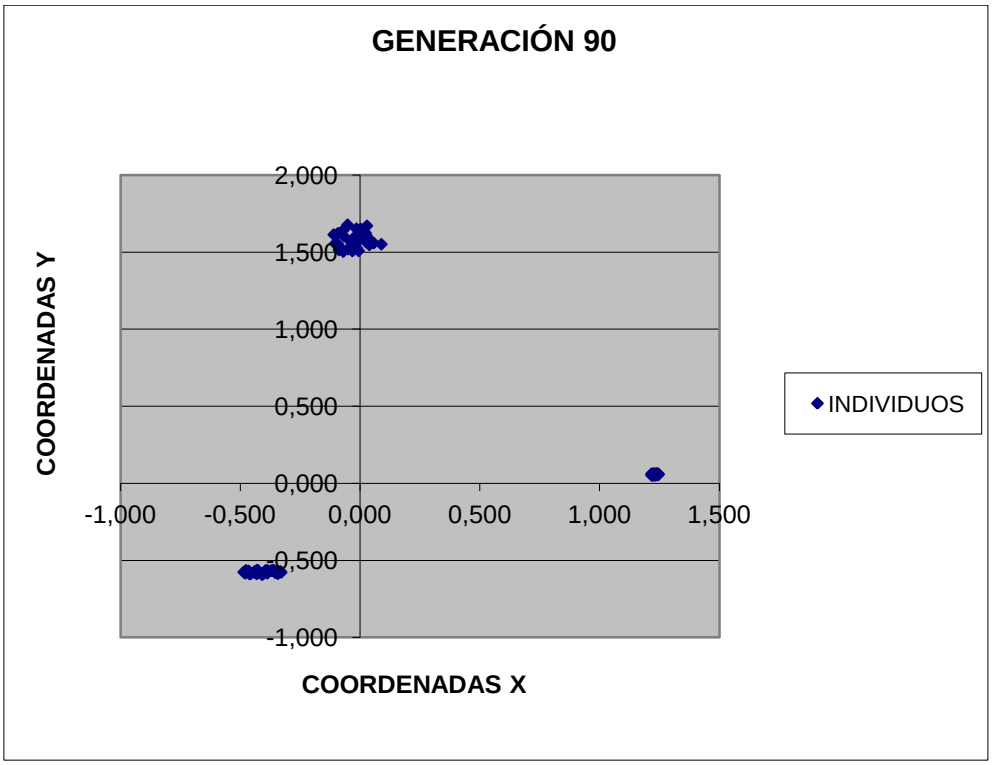


Fig 6.90 Localización de los individuos de la población en la generación 60

Para finalizar el estudio se realiza una toma de resultados una vez finalizado el algoritmo. Observando los datos de la tabla 6.10, los puntos se van agrupando en tres puntos en el plano.

GENERACIÓN FINAL					
NUM	VALOR X	VALOR Y	NUM	VALOR X	VALOR Y
0	-0,044	1,625	50	0,086	1,663
1	-0,005	1,574	51	-0,019	1,545
2	1,243	0,055	52	0,040	1,510
3	0,065	1,623	53	0,048	1,669
4	1,235	0,064	54	-0,087	1,632
5	-0,452	-0,562	55	-0,084	1,628
6	0,050	1,582	56	1,218	0,057
7	1,247	0,053	57	1,232	0,061
8	1,243	0,059	58	1,222	0,056
9	1,236	0,053	59	-0,438	-0,584
10	0,047	1,633	60	-0,394	-0,566
11	1,248	0,065	61	1,244	0,053
12	0,047	1,508	62	-0,366	-0,577
13	-0,026	1,552	63	-0,060	1,679
14	-0,046	1,559	64	-0,428	-0,569
15	-0,406	-0,581	65	-0,033	1,664
16	-0,348	-0,571	66	-0,071	1,669
17	-0,029	1,560	67	1,232	0,057
18	1,242	0,053	68	1,222	0,059
19	-0,386	-0,593	69	-0,088	1,598
20	-0,379	-0,582	70	0,007	1,572
21	0,036	1,507	71	1,226	0,054
22	-0,109	1,679	72	-0,481	-0,586
23	-0,411	-0,589	73	-0,469	-0,590
24	1,225	0,059	74	-0,470	-0,594
25	0,047	1,628	75	-0,067	1,553
26	0,027	1,517	76	-0,336	-0,591
27	1,227	0,061	77	-0,358	-0,592
28	1,241	0,058	78	1,239	0,056
29	1,244	0,062	79	1,234	0,053
30	1,235	0,062	80	1,246	0,054
31	1,244	0,056	81	1,248	0,057
32	0,050	1,531	82	1,246	0,053
33	1,228	0,065	83	-0,035	1,505
34	1,235	0,059	84	0,092	1,666
35	0,096	1,600	85	-0,327	-0,579
36	-0,344	-0,582	86	-0,448	-0,586
37	1,226	0,064	87	-0,438	-0,573
38	-0,342	-0,578	88	-0,421	-0,588
39	-0,348	-0,589	89	-0,399	-0,589
40	-0,445	-0,576	90	-0,407	-0,595
41	-0,386	-0,570	91	-0,375	-0,586
42	0,026	1,548	92	-0,479	-0,593
43	0,082	1,635	93	-0,440	-0,563
44	-0,105	1,585	94	-0,408	-0,570
45	1,248	0,063	95	1,222	0,059
46	1,225	0,058	96	-0,382	-0,583
47	-0,356	-0,598	97	1,229	0,064
48	-0,397	-0,567	98	1,245	0,064
49	-0,106	1,599	99	-0,027	1,671

Tabla 6.10 Datos de la generación finalizado de la función x.2.

Como se puede observar en las figura 6.91 todos los individuos de la población se siguen agrupando en la zona de las tres soluciones que se están buscando. Las soluciones obtenidas están alrededor de estos valores (0.000,1.590), (1.289,0.000) y (-0.460,-0.629). Y las soluciones reales de la función x.2 son las siguientes (-0.009,1.581), (1.286,-0.005) y (-0.460,-0.629).

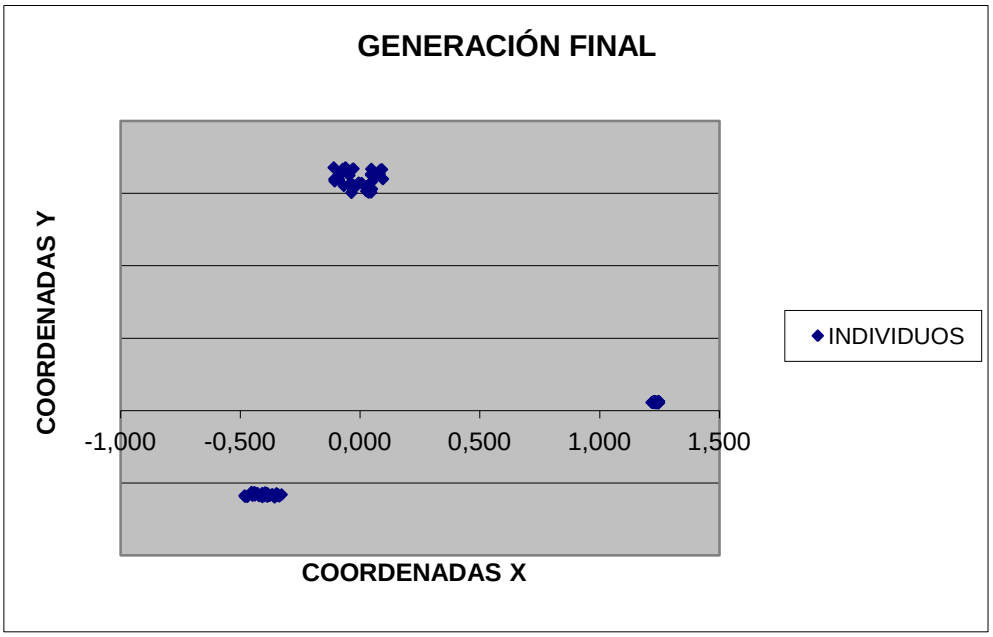


Fig 6.91 Localización de los individuos de la población en la final

Capítulo 7. Conclusiones y Futuros Trabajos

7.1 Conclusiones

Los AG´s son una fuerte fuente para la resolución de problemas del mundo real. Usan una analogía directa con el comportamiento natural. Trabajan con una población de individuos, cada uno de los cuales representa una solución factible a un problema dado. A cada individuo se le asigna un valor. Efectúa una búsqueda óptima, maximizando las ganancias o minimizando las perdidas. Una de las desventajas de los AG´s es que tienden a converger hacia una única solución y para muchos problemas se desean obtener diferentes soluciones al problema. En el presente trabajo se han estudiado diferentes técnicas para mantener la diversidad genética y obtener más de una solución. Incluso se ha implementado una librería para poder solucionar el problema de funciones multimodales.

Una de los principales problemas que se han encontrado al realizar la librería ha sido al programar. Los errores que se comenten son bastante difíciles de encontrar que en otro tipo de programas. Un error en la programación no suele provocar una caída del algoritmo. En ocasiones los fallos en el código se refleja en los resultados obtenidos, distorsionando las soluciones pero válido en todo caso como solución.

Al crear una librería para nichos, unos de los problemas a tratar es la convergencia prematura. La similitud genética de los individuos impide producir soluciones novedosas. Se pierde la capacidad de intercambio de información útil entre los individuos. La convergencia prematura hace que el proceso se estanque en máximo local. En la librería se ha intentado que exista una diversidad en la población en la mayoría de las fases de la evolución. En las últimas generaciones sí que se buscaba que los individuos perteneciesen a la zona de búsqueda de los nichos. Además no conviene que exista una convergencia prematura porque si se incrementan los individuos más aptos, disminuye la diversidad de la población y puede provocar que se produzca una avalancha a una única solución, siendo dificultoso encontrar el resto de soluciones. Sobre la diversidad de la población en las primeras fases de la evolución no se debería favorecer a los individuos más aptos, para obtener una mayor diversidad en la población, siendo menos eficiente las etapas de selección. Y en las últimas fases de la evolución se debería favorecer a los individuos más aptos para encontrar cuanto antes la solución.

Otro problema surge cuando la función de evaluación es muy abrupta, porque no existe suficiente precisión para aproximarse sucesivamente a los óptimos. No existe ningún indicador que dirija a los individuos a la solución. En este caso la búsqueda se parece más a un proceso aleatorio.

Una de las complicaciones de los AG´s es elegir correctamente la función de fitness, al ser lo único que incorpora conocimiento específico del problema a resolver. En nuestro caso era sencillo por qué consistía en maximizar o minimizar una función. Existen muchos problemas de optimización gran cantidad de restricciones, en los que buena parte de los puntos de búsqueda representa individuos no válidos siendo más complicado sacar los valores correctos.

Para terminar indicar que las técnicas de nicho es una alternativa importante para evitar la convergencia en óptimos locales, así como una herramienta importante en el desarrollo de AG para problemas multiobjetivo, obteniendo múltiples soluciones en una única ejecución manteniendo soluciones de diferentes parte del dominio.

7.2 Futuros Trabajos

En la actualidad uno de los puntos débiles en los AG's es su moderado o alto consumo de recursos computacionales (memoria y/o tiempo del procesador). Sobre problemas de elevada complejidad puede verse mejorada si se utilizasen sistemas distribuidos que se pudiesen ejecutar algoritmos genéticos en paralelos. Se puede realizar diferentes enfoques a la hora de utilizar sistemas distribuidos. El primero sería que cada procesador operara en una población aislada de individuos y que los mejores individuos de cada población aislada operarse con el resto de procesadores. Otro enfoque sería tener sólo una única población y que cada procesador realizase cada paso del algoritmo (selección, cruce, mutación). Y por último la tercera opción sería la combinación de la dos anteriores

Otra línea de trabajo que se puede estudiar es cómo los AG funcionan con problemas con restricciones y cómo deben ser tratados. O cómo los algoritmos resolverían un problema con múltiples funciones que deberían optimizarse, o al menos ser satisfechas simultáneamente. Como se comportarían en maximizar o minimizar un número de funciones para obtener un resultado óptimo. En este problema se debería buscar una buena función de fitness que pueda cumplir todo las condiciones que se indican. Y como se comportaría el AG con funciones de una alta complejidad. En este sentido, ya en la actualidad se han desarrollado y estudiado una gran diversidad de algoritmos, los cuales podrían ser mejorados. La librería desarrollada podría ser adaptada para abordar este tipo de problemas.

Capítulo 8. Presupuesto

A continuación se procede al desglose del presupuesto ligado al Proyecto Fin de Carrera. Estos costes provienen de gastos de personal, pero también de gastos de materiales y se puede ver en las siguientes tablas 8.1 y 8.2
En la tabla 8.1 se muestra las fases del proyecto y el tiempo aproximado para cada una ellas.

Fase	**Coste/Hora**	**Total horas**	**Coste de horas**
Documentación	35€	50	1,750€
Diseño	45€	45	2,025€
Codificación	40€	110	4,400€
Pruebas	30€	40	1,200€
Redacción del la memoria	40€	60	2,400€
		Total	**11,775€**

Tabla 8.1: Fase del Proyecto

En la tabla 8.2 se pueden ver los costes asociados a material, repartidos entre un ordenador donde se realiza el trabajo desarrollado y gastos varios como material fungibles, Internet, electricidad etc.

Concepto	**Coste**
Ordenador	875€
Gastos varios	450€
Total	**1,325€**

Tabla 8.2: Coste de Material

En la siguiente tabla 8.3 se encuentra el presupuesto total

Concepto	**Coste**
Fase del Proyecto	11,775€
Coste de Material	1,325€
Base imponible	13,100€
IVA(21%)	2,751€
Total	**15**,851€

El coste total del proyecto asciende a QUINCE MIL OCHOCIENTOS CICUENTA Y UN EUROS (15851€)

Bibliografía

Altshuler, Edward y Derek Linden. ``Design of a wire antenna using a genetic algorithm." Journal of Electronic Defense, vol.20, no.7, p.50-52 (julio de 1997)

Andre, David y Astro Teller. ``Evolving team Darwin United." En RoboCup-98: Robot Soccer World Cup II, Minoru Asada and Hiroaki Kitano (eds). Lecture Notes in Computer Science, vol.1604, p.346-352. Springer-Verlag, 1999.

Au, Wai-Ho, Keith Chan, y Xin Yao. ``A novel evolutionary data mining algorithm with applications to churn prediction." IEEE Transactions on Evolutionary Computation, vol.7, no.6, p.532-545 (diciembre de 2003).

Thomas Bäck. Optimization by Means of Genetic Algorithms. University of Dortmund, Germany 1991

Begley, Sharon y Gregory Beals. ``Software au naturel." Newsweek, 8 de mayo de 1995, p.70.

Benini, Ernesto y Andrea Toffolo. ``Optimal design of horizontal-axis wind turbines using blade-element theory and evolutionary computation." Journal of Solar Energy Engineering, vol.124, no.4, p.357-363 (noviembre de 2002).

Brindle A.Genetic Algorithms for Function Optimization.PhD thesis, Department of Computer Science, University of Alberta, Edmonton, Alberta, (1981)

Burke, E.K. y J.P. Newall. ``A multistage evolutionary algorithm for the timetable problem." IEEE Transactions on Evolutionary Computation, vol.3, no.1, p.63-74 (abril de 1999).

Charbonneau, Paul. ``Genetic algorithms in astronomy and astrophysics." The Astrophysical Journal Supplement Series, vol.101, p.309-334 (diciembre de 1995).

Chellapilla, Kumar y David Fogel. ``Evolving an expert checkers playing program without using human expertise." IEEE Transactions on Evolutionary Computation, vol.5, no.4, p.422-428 (agosto de 2001).

Chryssolouris, George y Velusamy Subramaniam. ``Dynamic scheduling of manufacturing job shops using genetic algorithms." Journal of Intelligent Manufacturing, vol.12, no.3, p.281-293 (junio de 2001).

C.A. Coello, D.A. Van Veldhuizen, G.B. Lamont, Evolutionary algorithmis for solving multi-objective problems. Kluwer Academic Publishers, 2002.
Coale, Kristi. ``Darwin in a box." Wired News, 14 de julio de 1997

C.Darwin. “The origin of species”,1859

David. E. Goldberg, K. Deb, and B. Korb. Don't worry, be messy. In Proc. Of the Fourth International Conference on Genetic Algorithms, pages 24-30, San Diego, CA, 1991.

David E.Goldber(1989). Genetic Algorthms in Search. Optimization & Machine Learning. Addison-Wesley Co,. Inc Reading, MA

DeJong. K(1985) Genetic Algorithms A 10 year perspective. In Proceedings of the First International Conference on Genetic Algorithms(p169-177). Lawrence Erlbaum Associates._Gibbs, W. Wayt. ``Programming with primordial ooze." Scientific American, octubre de 1996, p.48-50.

Gillet, Valerie. ``Reactant- and product-based approaches to the design of combinatorial libraries." Journal of Computer-Aided Molecular Design, vol.16, p.371-380 (2002).

Giro, R., M. Cyrillo y D.S. Galvao. ``Designing conducting polymers using genetic algorithms." Chemical Physics Letters, vol.366, no.1-2, p.170-175 (25 de noviembre de 2002).

Glen, R.C. y A.W.R. Payne. ``A genetic algorithm for the automated generation of molecules within constraints." Journal of Computer-Aided Molecular Design, vol.9, p.181-202 (1995).

Goldberg, D.E .& Deb, K."A comparative analysis on selection schemes used in genetic algorithms".Foundations of Genetic Algorithms 1. Rawlins G.(Ed). Kaufmann. San Mateo.Pag. 69-93. (1991).

Grosso. P.B(1985) Computer simulations of genetic adaptation: Parallel subcomponent interaction in a multilocus model . Tesis. The University o fMichigan Ann Arbor.

Haas, O.C.L., K.J. Burnham y J.A. Mills. ``On improving physical selectivity in the treatment of cancer: A systems modelling and optimisation approach." Control Engineering Practice, vol.5, no.12, p.1.739-1.745 (diciembre de 1997).

Haupt, Randy y Sue Ellen Haupt. Practical Genetic Algorithms. John Wiley & Sons, 1998.

He, L. y N. Mort. ``Hybrid genetic algorithms for telecommunications network back-up routeing." BT Technology Journal, vol.18, no.4, p. 42-50 (octubre de 2000).
Hillier,F.S y Lieberman, G.J (1986): Introduccion a la Investigación de Operaciones. MCGraw-Hill.

W.D.Hillis, "Coevolving Parasites Improve Simulated Evolution as an Optimization Procedure," Physica D, vol 42 pp 228-234 Jun 1990
John H. Holland (1996) Adaption innatural and aritificial systems. The University of Michigan press; ann Arbor,1975

Jolley L.B.W(1961) Summation of series Dover

Juille H.(1993) Coevolutinary learning a case study. Proceedin of the fittenth international conference on machine learning. Morgan Kauffmann (pag251-259)

Juille and J.B. Pollack. Advances in Genetic Programming 2, chapter Massively parallel genetic programming, The MIT Press, MA, USA, 1996.

Hughes, Evan y Maurice Leyland. ``Using multiple genetic algorithms to generate radar point-scatterer models." IEEE Transactions on Evolutionary Computation, vol.4, no.2, p.147-163 (julio de 2000).

Jensen, Mikkel. ``Generating robust and flexible job shop schedules using genetic algorithms." IEEE Transactions on Evolutionary Computation, vol.7, no.3, p.275-288 (junio de 2003).

Keane, A.J. y S.M. Brown. ``The design of a satellite boom with enhanced vibration performance using genetic algorithm techniques." En Adaptive Computing in Engineering Design and Control '96 - Proceedings of the Second International Conference, I.C. Parmee (ed), p.107-113. University of Plymouth, 1996.

Kewley, Robert y Mark Embrechts. ``Computational military tactical planning system." IEEE Transactions on Systems, Man and Cybernetics, Part C - Applications and Reviews, vol.32, no.2, p.161-171 (mayo de 2002).

Koza, John, Forest Bennett, David Andre y Martin Keane. Genetic Programming III: Darwinian Invention and Problem Solving. Morgan Kaufmann Publishers, 1999.

Lee, Yonggon y Stanislaw H. Zak. ``Designing a genetic neural fuzzy antilock-brake-system controller." IEEE Transactions on Evolutionary Computation, vol.6, no.2, p.198-211 (abril de 2002).

Mahfoud, Sam y Ganesh Mani. ``Financial forecasting using genetic algorithms." Applied Artificial Intelligence, vol.10, no.6, p.543-565 (1996).
Michalewicz, Z. (1996). Genetic Algorithms + Data Structures = Evolution Programs.Springer Verlag. 3 º Edición.

Mitchell, Melanie. An Introduction to Genetic Algorithms. MIT Press, 1996.
Naik, Gautam. ``Back to Darwin: In sunlight and cells, science seeks answers to high-tech puzzles." The Wall Street Journal, 16 de enero de 1996, p. A1.

Obayashi, Shigeru, Daisuke Sasaki, Yukihiro Takeguchi, y Naoki Hirose. ``Multiobjective evolutionary computation for supersonic wing-shape optimization." IEEE Transactions on Evolutionary Computation, vol.4, no.2, p.182-187 (julio de 2000).

Paradis, Perrien, J y P.M Banting(1995) “Dissolution of a relationship: the salesforce perception”, Industrial Marketing Management” Vol. 24, n.4

Rechenberg I. Evolutionsstrategie: Optimierung Technischer Systeme Nach Prinzipien Der Biologischen Evolution , Frommann-Holzboog Verlag, Stuttgart, 1973

Rizki, Mateen, Michael Zmuda y Louis Tamburino. ``Evolving pattern recognition systems." IEEE Transactions on Evolutionary Computation, vol.6, no.6, p.594-609 (diciembre de 2002)

Robin, Franck, Andrea Orzati, Esteban Moreno, Otte Homan, y Werner Bachtold. ``Simulation and evolutionary optimization of electron-beam lithography with genetic and simplex-downhill algorithms." IEEE Transactions on Evolutionary Computation, vol.7, no.1, p.69-82 (febrero de 2003)

Sambridge, Malcolm y Kerry Gallagher. ``Earthquake hypocenter location using genetic algorithms." Bulletin of the Seismological Society of America, vol.83, no.5, p.1.467-1.491 (octubre de 1993).

Sasaki, Daisuke, Masashi Morikawa, Shigeru Obayashi y Kazuhiro Nakahashi. ``Aerodynamic shape optimization of supersonic wings by adaptive range multiobjective genetic algorithms." En Evolutionary Multi-Criterion Optimization: First International Conference, EMO 2001, Zurich, Switzerland, March 2001: Proceedings,

K. Deb, L. Theile, C. Coello, D. Corne y E. Zitler (eds). Notas de la confenrencia en Computer Science, vol.1993, p.639-652. Springer-Verlag, 2001.

Sato, S., K. Otori, A. Takizawa, H. Sakai, Y. Ando y H. Kawamura. ``Applying genetic algorithms to the optimum design of a concert hall." Journal of Sound and Vibration, vol.258, no.3, p. 517-526 (2002).

Schechter, Bruce. ``Putting a Darwinian spin on the diesel engine." The New York Times, 19 de septiembre de 2000, p. F3.

Schwefel H.P. Evolution Strategies: A Family of Non Linear Optimization Techniques Based on Imitating Some Principles of Organic Evolution. Annals of Operations Research volumen 1, 1984: 165-167.

Smith. R.E Forrest, S & Perelson A.S(1992) "Searching for diverse, cooperative populations with genetic algorithms". TCGA Report No 92002 The University of Alabama Department of Engineering Mechanics

Tang, K.S., K.F. Man, S. Kwong y Q. He. ``Genetic algorithms and their applications." IEEE Signal Processing Magazine, vol.13, no.6, p.22-37 (noviembre de 1996).

Weismann, Dirk, Ulrich Hammel, y Thomas Bäck. ``Robust design of multilayer optical coatings by means of evolutionary algorithms." IEEE Transactions on Evolutionary Computation, vol.2, no.4, p.162-167 (noviembre de 1998).

Williams, Edwin, William Crossley y Thomas Lang. ``Average and maximum revisit time trade studies for satellite constellations using a multiobjective genetic algorithm." Journal of the Astronautical Sciences, vol.49, no.3, p.385-400 (julio-septiembre de 2001).