UNIVERSIDAD CENTROCCIDENTAL
"LISANDRO ALVARADO"
DECANATO DE CIENCIAS
MAESTRÍA EN CIENCIAS MENCIÓN OPTIMIZACIÓN

UN ALGORITMO EVOLUTIVO PARA MINIMIZAR UNA FUNCIÓN DE ORDEN P

Autor: Lcdo. Adrián Rojas
Tutor: Dr. Rómulo Castillo

Barquisimeto, enero de 2011

UNIVERSIDAD CENTROCCIDENTAL
"LISANDRO ALVARADO"
DECANATO DE CIENCIAS
MAESTRÍA EN CIENCIAS MENCIÓN OPTIMIZACIÓN

UN ALGORITMO EVOLUTIVO PARA MINIMIZAR UNA FUNCIÓN DE ORDEN P

Trabajo presentado para optar al Grado de
Magister Scientiarum. Mención Optimización

Autor: Lcdo. Adrián Rojas
Tutor: Dr. Rómulo Castillo

Barquisimeto, enero de 2011

Agradecimientos

- Primeramente a mi Dios por darme la sabiduría y la voluntad por mejorar como persona, agradecido estoy mi Dios!!!.
- Al Dr. Rómulo Castillo Cárdenas, por ser, además de mi tutor, un amigo que siempre me mostró su ayuda. Muchísimas gracias Profesor!
- Al Dr. Hugo Lara y la Dra. Flor Montes de Oca, ya que ellos formaron parte de mi formación académica.
- A Clavel Quintana, por brindarme su ayuda en todo momento. Muchas gracias !!!
- A Juan Carlos Juarez por su grandiosa ayuda en los momentos. Gracias por todo amigo mio!!.
- A Karmela, Elena y Henry por su apoyo y muestra de gentileza; Gracias por estar en esos momentos en que les necesite.
- A mis amigos: Ernesto López, Elvis Aponte, José Mogollón y Dewis Gómez, por acompañame en todo ese tiempo de estudio.
- A la Señora Auxiliadora, por tenerme en su casa como un hijo. Mil gracias por todo su aprecio!!!.
- A la UCLA por ser el lugar donde recibí mi formación profesional.
- A todas aquellas personas que de una u otra forma son parte de este trabajo.

A mi **Padre Celestial, Dios**, sobre todas las cosas.

A mis padres: **Ana Díaz y Luciano Rojas**, que me brindan su amor y apoyo todo momento.

A mis **hermanos y sobrinos**, que me dieron todo lo que pudieron darme.

A mi hija **Andrea**, por ser el motivo de mi vida.

A **Steven y Marisol**, por ser parte de mi familia.

A mi esposa **Marbelis Rodríguez**, por todo su amor, comprensión y gran apoyo.

Resumen

El problema OVO (Order-Value Optimization) es un problema de optimización introducido recientemente, en el cual, dadas m funciones reales con un dominio común, se desea encontrar el minimizador de la función que ocupe el lugar p, con p menor o igual a m. En el caso en que p sea igual a m, tenemos el clásico problema minimax. Este problema ha sido considerado en aplicaciones financieras, estimación robusta y análisis de datos y ha sido resuelto mediante técnicas de optimización continua como por ejemplo, método de Cauchy y método Casi-newton, siendo su principal desventaja la dificultad de encontrar minimizadores globales en algunas circunstancias. Por otra parte, los algoritmos evolutivos constituyen herramientas de optimización y búsqueda de soluciones basadas en postulados de evolución biológica y han sido de gran uso en diversos problemas donde los algoritmos de optimización continua no son tan eficaces. En el presente trabajo resolvemos el problema OVO, para ello se estudian las caracteristicas y propiedades teóricas de dicho problema y luego se diseña e implementa un algoritmo evolutivo en matlab.

Palabras claves: Optimización continua, algoritmos evolutivos, problema OVO.

Introducción

En la optimización podemos escribir de manera general el problema $min_{x\in\Omega} f(x)$ donde $f : \Omega \subset \mathbb{R}^n \to \mathbb{R}$ es la función objetivo, un caso particular de este problema es la conocida $f(x) = min_{x\in\mathbb{R}^n} max f_i(x) : i \in I$ donde f_i para $i = 1, ..., k$ son funciones dadas.
El problema de minimización de una función de orden p lo describimos a continuación:
Dadas m funciones continuas $f_1, ..., f_m$, definidas en un dominio $\Omega \in \mathbb{R}^n$ y un entero $p \in \{1, ..., m\}$, la funcion de p orden está dada por $f(x) = f_{i_p(x)}(x)$, para toda $x \in \Omega$, donde $p(x)$ es una funcion índice que satisface

$$f_{i_1(x)}(x) \leq f_{i_2(x)}(x) \leq ... \leq f_{i_p(x)}(x) \leq ... \leq f_{i_m(x)}(x).$$

El problema que trataremos es entonces

$$\text{Minimizar } f(x), x \in \Omega.$$

Este problema es denominado por sus primeros proponentes como problema OVO por su nombre en inglés (Order Value Optimization) e involucra la minimización de una función que en general es continua pero no es suave aún en el caso en que las funciones involucradas sean diferenciables, observe que para $p = m$ obtenemos el clásico ploblema min-max. El problema OVO fue introducido recientemente en [2],[4],[3] y [5] en donde además de hacer una formulación más general como un problema de programación no-lineal con restricciones, introducen algunas formas de resolverlo usando métodos de Cauchy y del tipo Casi-Newton. La principal motivación para este problema data, entre otras, de aplicaciones en problemas de toma de decisiones y problemas robustos de estimacion de parámetros.
Por otra parte, conocemos que algoritmos genéticos y evolutivos han sido usados con eficiencia en diversidad de problemas, [16], [14], [15] y [12], principalmente cuando no se conocen métodos eficientes para su resolución.

Este es el caso del problema OVO que por ser de reciente estudio, los métodos usados estan en constante experimentación y de aqui nuestra propuesta. Los algoritmos evolutivos constituyen técnicas computacionales basadas en heurísticas y fundamentos probabilísticos y deben su

nombre a comparaciones biológicas de reproducción y adaptación de las especies vivientes al campo computacional. se parte de una población inicial de la cual se seleccionan hijos, éstos se reproducen bajo ciertos criterios y se efecúa una mutación. Estos nuevos individuos se evaluán y se incorporan a una nueva población que constituye otro iterado del algoritmo evolutivo. La técnica para la escogencia de la población, su reproducción y mutación serán objeto de estudio en este trabajo para así escoger la que más se adapte a nuestro problema.

El problema consiste finalmente en aplicar las técnicas de algoritmos evolutivos para resolver el problema OVO.

En el capitulo I, se muestra un breve resumen de la teoria que usaremos en los capitulos posteriores, definiendo en primer lugar los fundamentos básicos de los algoritmos evolutivos: concepto de computación evolutiva, codificación o representación de la población seguidamente se estudia las técnicas de selección, los operadores evolutivos o de reproduccion: técnicas de cruza y mutación y así ofrecer una mejor comprensión de esta investigación.

En el capitulo II, se estudia el problema OVO donde se hace un estudio de dicho problema utilizando métodos de Cauchy y del tipo Casi-Newton para resolver problemas de programación no lineal con restricciones.

Finalmente en el capitulo III, se presenta el aporte original de este trabajo que es la elaboración e implementación en matlab del algoritmo evolutivo para minimizar una función de orden p.

Índice general

Capítulo 1

Algoritmos Evolutivos

A continuación se dará un breve resumen referente a la teoria de algoritmos evolutivos, extraida mayormente de [12] y [14].

Los algoritmos evolutivos basan su funcionamiento en la simulación del proceso de evolución natural (Goldberg, 1989). El proceso de evolución puede ser visto como una búsqueda entre el conjunto enorme de posibles secuencias de genes de aquellas que correspondan a los mejores individuos. Esta búsqueda que la naturaleza realiza es robusta, en el sentido que es capaz de localizar eficaz y eficientemente individuos de mejor adaptación en un espacio de búsqueda enorme y cambiante.

Las reglas de la evolución son notablemente simples. Las especies evolucionan por medio de la selección natural, esto es, la conservación de las diferencias y variaciones individualmente favorables y la destrucción de las que son perjudiciales. Los individuos que tienen ventaja sobre otros, por ligera que sea, tendrán más probabilidades de sobrevivir y procrear su especie.

Los hijos son semejantes a sus padres, por lo que cada nueva generación tiene individuos semejantes a los mejores individuos de la generación anterior. Además del proceso de selección natural, en la naturaleza ocasionalmente se producen mutaciones al azar. Algunas de estas mutaciones conducirán a nuevos y mejores individuos, de esta manera, alteraciones aleatorias serán introducidas y luego propagadas a generaciones futuras. Por el contrario, las características negativas introducidas por la mutación tenderán a desaparecer pues conducen a individuos pobremente adaptados. Si las modificaciones del ambiente son menores, las especies irán evolucionando gradualmente con éste; sin embargo, es probable que un súbito cambio de ambiente provoque la desaparición de especies enteras y el nacimiento de otras nuevas, capaces de sobre-

vivir en el nuevo entorno.

Los algoritmos evolutivos son algoritmos de búsqueda y optimización que simulando los mecanismos esenciales de la evolución Darwiniana, intentan reproducir las características de robustez y simplicidad existentes en la naturaleza para evolucionar hacia mejores soluciones en problemas computacionales.

Consisten en una técnica iterativa que aplica operadores estocásticos sobre un conjunto de individuos (la población) con el propósito de mejorar su fitness, una medida relacionada con la función objetivo del problema en cuestión. Cada individuo de la población representa una solución potencial del problema, codificada de acuerdo a un esquema de representación, generalmente basado en números binarios o reales.

Inicialmente la población se genera de forma aleatoria, y luego evoluciona mediante la aplicación iterativa de interacciones denominadas operadores de reproducción, que incluyen recombinaciones de individuos (los cruzamientos) y modificaciones aleatorias (las mutaciones). Esta evolución es guiada por una estrategia de selección de los individuos más adaptados a la resolución del problema, de acuerdo a sus valores de fitness.

En términos generales, para simular el proceso evolutivo en una computadora se requiere:

- Codificar las estructuras que se replicarán.
- Operaciones que afecten a los “individuos ”(cruce y mutación).
- Una función aptitud.
- Un mecanismo de selección.

Existen muchos problemas que tienen alta dimensionalidad, son discontinuos, multimodales y/o NP-completos.
Los problemas que poseen una o más de estas características los denominaremos irregulares. Si bien los métodos determinísticos han sido utilizados exitosamente en la resolución de una gran variedad de problemas, son usualmente inefectivos cuando se aplican a problemas irregulares.
De acuerdo al no-free-lunch theorem (NFL), no existe ningún algoritmo para resolver todos los problemas de optimización que sea en promedio superior a cualquier buen competidor. Sin embargo, los algoritmos evolutivos han demostrado poseer características que les permiten presentar un mejor comportamiento que el demostrado por otros métodos para resolver un gran número de

problemas reales donde las técnicas clásicas no pueden ser aplicados, aunque como consecuencia, se comportan peor para otra clase de problemas. En particular, los métodos clásicos son muy eficientes para la resolución de problemas lineales o cuadráticos, fuertemente convexos y unimodales. Por otra parte, los algoritmos evolutivos pueden ser utilizados en problemas discontinuos, no diferenciables, multimodales, ruidosos y en otras superficies de búsqueda poco convencionales donde la búsqueda por métodos clásicos es impracticable. Entonces, su efectividad se extiende a un número mayor de aplicaciones, por supuesto, con una pérdida de eficiencia cuando se aplican a problemas para los cuales se tiene un método específico para resolverlo. En este sentido, se puede decir que los AEs (Algoritmos Evolutivos) son más robustos que otras alternativas. Las características generales que hacen a los AEs diferentes a los métodos tradicionales en cuanto a la robustez son:

1. Trabajan con poblaciones de individuos donde cada individuo representa (o codifica) un punto en el espacio de soluciones potenciales para un problema dado. Esto implica que trabajan con muchos puntos reduciendo de ésta forma la posibilidad de perderse en óptimos locales.

2. Las soluciones se generan de forma iterativa mediante la utilización de operadores aleatorios que intentan modelar los procesos esenciales que intervienen en la evolución natural. Existen tres operadores evolutivos principales: cruzamiento, mutación y selección.

3. A fin de evaluar los individuos se utiliza información de su función objetivo en forma directa para asignarle un valor de adaptación que permite determinar si un individuo es mejor que otro.

La diferencia básica existente entre los diferentes enfoques de la computación evolutiva radica en la manera en que los principios evolutivos son utilizados para obtener las características anteriores. Entre los diferentes modelos evolutivos principales se tiene que:

- Los algoritmos genéticos (AGs) enfatizan el operador de cruzamiento como el operador de búsqueda más importante y aplican la mutación con una pequeña probabilidad como un operador de segundo plano. Los GAs también utilizan un operador de selección probabilístico y usualmente utilizan una representación binaria de los individuos.

  Algunas aplicaciones de los AGs son las siguientes :

  - Optimización (estructural, de topologías, numérica, combinatoria, etc.)
  - Aprendizaje de máquina (sistemas clasificadores)

- Bases de datos (optimización de consultas)
- Reconocimiento de patrones (por ejemplo, imágenes)
- Generación de gramáticas (regulares, libres de contexto, etc.)
- Planeación de movimientos de robots
- Predicción

- Las estrategias de evolución utilizan mutaciones con distribución normal para modificar vectores de valores reales enfatizando la mutación y el cruzamiento como sus operadores principales para la exploración del espacio de búsqueda. El operador de selección es determinístico y las poblaciones padre e hijo usualmente difieren en tamaño.

  Algunas aplicaciones de las estrategias evolutivas son :

  - Problemas de ruteo y redes
  - Bioquímica
  - Óptica
  - Diseño en ingeniería
  - Magnetismo

- La programación evolutiva enfatiza la mutación y no incorpora la recombinación de individuos. Al igual que las estrategias de evolución, utilizan distribución normal para su operador de mutación. El operador de selección es probabilístico, y aunque en la actualidad la mayor parte de sus aplicaciones son para búsquedas en espacios de vectores de valor real, el algoritmo fue desarrollado en un principio para evolucionar máquinas de estado finito.

  Algunas aplicaciones de la programación evolutiva son :

  - Predicción
  - Generalización
  - Juegos
  - Control automático
  - Problema del viajero
  - Planeación de rutas

- Diseño y entrenamiento de redes neuronales
- Reconocimiento de patrones

1.1. Conceptos de Computacion Evolutiva

En general, para resolver un problema utilizando una computadora, primeramente buscamos la secuencia de pasos que deben ejecutarse para resolverlo (el algoritmo). Luego, utilizando algún lenguaje de programación, se codifica esta secuencia de pasos en un conjunto de instrucciones que se han de ejecutar. Cuando se trabaja con algoritmos evolutivos, sin embargo, la preocupación principal no es acerca de qué pasos conducirán a la solución del problema sino más bien en la manera de representar las soluciones. Es el algoritmo evolutivo el que se encargará de evolucionar la solución. Así, la computación evolutiva representa un cambio radical con respecto a la manera tradicional de enfrentar los problemas computacionales.

Para resolver un problema utilizando técnicas de computación evolutiva, las posibles soluciones son codificadas de alguna manera similar a la estructura de las cadenas de genes que conforman los cromosomas. Los parámetros del problema se codifican ya sea como una cadena de números binarios , números reales , letras del alfabeto, listas, etc. Cada parámetro puede tener una codificación independiente de acuerdo a la conveniencia. En analogía con la naturaleza, cada parámetro del problema codificado representa un gen y considerados en conjunto, forman una cadena de genes o cromosoma.
Así mismo, los elementos en el conjunto de parámetros son denominados usualmente genotipos, mientras que los elementos en el espacio objetivo suelen denominarse fenotipos.

Definición 1.1 Codificacion: Sea S un espacio de búsqueda, i.e. una colección de objetos sobre el cual se realizará la búsqueda. Sean $A_1, A_2, ..., A_n$ conjuntos finitos arbitrarios y sea:

$$I = A_1 \times A_2 \times, ..., \times A_n$$

Finalmente sea

$$g : I \longrightarrow S$$

una función que mapea vectores en I a soluciones en el espacio de búsqueda. Se dice que el par $(I, g : I \longrightarrow S)$ es una codificación o representación de S. El conjunto I es denominado espacio de codificación, la función g es llamada la función de mapeo. Entonces, n es el número de variables utilizadas para mapear una solución de S en I.

Definición 1.2 Alelo: Los conjuntos, $A_i, i \in 1, 2, ..., n$ a partir de los cuales se compone I en la definición 2.1, es llamado el conjunto de alelos y sus miembros alelos.

Definición 1.3 Cromosoma: Sea la codificación $(I, g : I \longrightarrow S)$, los miembros de I son llamados cromosomas y menos comúnmente genomas o genotipos. Un cromosoma $\mathbf{x} \in I$ puede ser escrito como:

$$(x_1, x_2, ..., x_n) \in I = A_1 \times A_2 \times, ..., \times A_n$$

o como una cadena $x_1 x_2 ... x_n$.

Definición 1.4 Gen: Los componentes x_i de $\mathbf{x}$, cuando se tratan como variables, son conocidos como genes, tal que el i-ésimo gen toma sus valores del conjunto de alelos A_i.

Definición 1.5 Locus: La posición i de un gen en el cromosoma es conocida como su locus.

Como ejemplo, considerando el espacio de búsqueda $S = 0, 1, 2, ..., 9$ y un espacio de representación $I = A_1 \times A_2$, donde $A_1 = 0, 1$ y $A_2 = 0, 1, 2, 3, 4$. Una función de mapeo puede definirse adecuadamente como: $g(x_1, x_2) = 5x_1 + x_2$ donde x_1 toma sus valores del conjunto de alelos A_1 y x_2 toma sus valores del conjunto de alelos A_2. Entonces, para la codificación $(I, g : I \longrightarrow S)$, un cromosoma $x = x_1 x_2 = 13$ en I representa a la solución 8 en S. Además, se dice que el gen con locus 1 (x_1), tiene como valor el alelo 1 y el gen con locus 2 tiene como valor el alelo 3.

En muchas representaciones, todos los genes tienen la misma cardinalidad y comparten un mismo conjunto de alelos. En la mayoría de las aplicaciones de Algoritmos Geneticos, las variables de decisión son codificadas utilizando representaciones binarias, en las cuales todos los genes tienen los alelos 0 o 1. Esto es, $A_i = 0, 1$ para todo $i \in 1, 2, ..., n$.
La representación binaria es adecuada para los problemas llamados problemas de optimización combinatoria seudo-boléanos de la forma $f : \{0, 1\}^l \longrightarrow \mathbb{R}$. En éstos problemas, el espacio de búsqueda $(\{0, 1\}^l)$ puede ser representado en forma directa por cadenas binarias de longitud $l = n$. Es decir, para este caso se tiene $S = I$. Algunos ejemplos de tales problemas son el problema del conjunto independiente máximo en grafos o el problema de la mochila, los cuales pueden ser representados por vectores binarios simplemente incluyendo (o excluyendo) un vértice o item i en (de) una solución candidata cuando la entrada correspondiente se hace $x_i = 1 (x_i = 0)$.

Esta representación también ha sido muy utilizada en el caso de problemas $f : S \longrightarrow \mathbb{R}$ donde el espacio de búsqueda S es fundamentalmente distinto del espacio vectorial binario $\{0,1\}^l$. Uno de los ejemplos más prominentes de esto está dado por la aplicación de algoritmos genéticos para problemas de optimización de parámetros continuos de la forma $f : \mathbb{R}^n \longrightarrow \mathbb{R}$. Los mecanismos para codificar y decodificar entre los dos espacios diferentes restringen el espacio continuo a intervalos finitos $[u_i; v_i]$ para cada variable, dividiendo el vector binario de tamaño l en n segmentos. Por lo general estos segmentos tienen igual longitud l_x, tal que $l = n.l_x$. Siendo a_j el valor (0 o 1) específico en la posición j de la cadena binaria, se interpreta el subsegmento ($a_{(i-1).l+1}, ..., a_{i.l}$) ($i = 1, ..., n$) como la codificación binaria de la variable x_i. Entonces, la decodificación del i-ésimo gen representado en forma binaria con una subcadena de tamaño $l_i = l_x$ se realiza de acuerdo a la función de decodificación binaria $\Gamma^i : \{0,1\}^l \longrightarrow [u_i; v_i]$, donde :

$$\Gamma^i(a_{i,1}, ..., a_{i.l}) = u_i + \frac{v_i - u_i}{2^l - 1}.(\sum_{j=0}^{l_x - 1} a_{i.l_x - j}.2^j) \quad (2.1)$$

Por ejemplo, una variable real x_1 en el rango $[0.5, 16]$ puede ser codificada en cadenas de 5 bits. Entonces, utilizando la ecuación 2.1, las cadenas

0	0	0	0	0

y

1	1	1	1	1

representan los valores reales 0.5 y 16 respectivamente, mientras que cualquiera de las otras 30 cadenas posibles representan una solución en el intervalo $[0.5, 16]$:

$$\begin{aligned}
\Gamma^1(00000) &= 0.5 + \frac{16 - 0{,}5}{2^5 - 1}.0 = 0.5 \\
\Gamma^1(00001) &= 0.5 + \frac{16 - 0.5}{2^5 - 1}.(1.2^0) = 1.0 \\
\Gamma^1(00010) &= 0.5 + \frac{16 - 0.5}{2^5 - 1}.(1.2^1 + 0.2^0) = 1.5 \\
&\vdots \\
\Gamma^1(11111) &= 0.5 + \frac{16 - 0.5}{2^5 - 1}.31 = 16.0
\end{aligned}$$

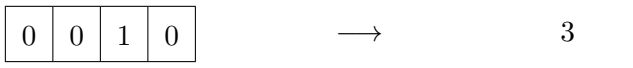
Definición 1.6. Genotipo: Se denomina genotipo a la codificación (por ejemplo, binaria) de los parámetros que representan una solución del problema a resolverse (ver figura 1.1).

1	0	1	1	0	0	1	0	0

Figura 1.1: Un ejemplo de una cadena cromosómica

Definición 1.7. Fenotipo: Se denomina fenotipo a la decodificación del cromosoma. Es decir, a los valores obtenidos al pasar de la representación (binaria) a la usada por la función

objetivo (ver figura 1.2).



Genotipo decodificación fenotipo

Figura 1.2:

Definición 1.8. Individuo Sea la codificación $(I, g : I \longrightarrow S)$, y sea el cromosoma $x \in I$, se utiliza el término individuo para referenciar una estructura de datos compuesta por un cromosoma y otras propiedades.

Definición 1.9. Población Una población es una estructura de datos que representa un grupo de individuos. Generalmente se representa a una población por medio de un vector de individuos.

Definicion 1.10. Aptitud: Se denomina aptitud al valor que se asigna a cada individuo y que indica qué tan bueno es éste con respecto a los demás para la solución de un problema. Por ejemplo, si $f(x) = x^2$, entonces $f(1010_2) = 100$ (donde $f(x)$ es la aptitud de un individuo).

Definicion 1.11. Generación Llamamos generación a una iteración de la medida de aptitud y a la creación de una nueva población por medio de operadores de reproducción.

Definicion 1.12. Operador de Reproducción Se llama operador de reproducción a todo aquel mecanismo que influencia la forma en que se pasa la información genética de padres a hijos. Los operadores de reproducción caen en tres amplias categorías:

- Cruza
- Mutación
- Reordenamiento

Definicion 1.13. Cruza La cruza es un operador que forma un nuevo cromosoma combinando partes de cada uno de sus cromosomas padres.

Definicion 1.14. Mutacion Se denomina mutación a un operador que forma un nuevo cromosoma a través de alteraciones (usualmente pequeñas) de los valores de los genes de un solo

cromosoma padre.

Definicion 1.15. Reordenamiento Un operador de reordenamiento es aquél que cambia el orden de los genes de un cromosoma, con la esperanza de juntar los genes que se encuentren relacionados, facilitando así la producción de bloques constructores.

Definicion 1.16. Elitismo Se denomina elitismo al mecanismo utilizado en algunos Algoritmos Evolutivos para asegurar que los cromosomas de los miembros más aptos de una población se pasen a la siguiente generación sin ser alterados por ningún operador genético.

Usar elitismo asegura que la aptitud máxima de la población nunca se reducirá de una generación a la siguiente. Sin embargo, no necesariamente mejora la posibilidad de localizar el óptimo global de una función. No obstante, es importante hacer notar que se ha demostrado que el uso de elitismo es vital para poder demostrar convergencia de un algoritmo genético.

1.2. Codificación o representación

Las capacidades para procesamiento de datos de las técnicas de computación evolutiva dentro de una amplia gama de dominios han sido reconocidas en los últimos años y han recibido mucha atención por parte de científicos que trabajan en diversas disciplinas. Dentro de estas técnicas evolutivas, quizás la más popular sea el algoritmo genético (AG). Siendo una técnica heurística estocástica, el algoritmo genético no necesita información específica para guiar la búsqueda. Su estructura presenta analogías con la teoría biológica de la evolución, y se basa en el principio de la supervivencia del más apto . Por lo tanto, el AG puede verse como una "caja negra "que puede conectarse a cualquier aplicación en particular. En general, se necesitan los cinco componentes básicos siguientes para implementar un AG que resuelva un problema cualquiera:

1. Una representación de soluciones potenciales al problema.
2. Una forma de crear una población inicial de soluciones potenciales (esto se efectúa normalmente de manera aleatoria, pero también pueden usarse métodos determinísticos).
3. Una función de evaluación que juega el papel del ambiente, calificando a las soluciones producidas en términos de su "aptitud ".
4. Operadores genéticos que alteran la composición de los descendientes (normalmente se usan la cruza y la mutación).

1	0	0	1	1	0	1

Figura 1.3: Un ejemplo de una cadena binaria.

5. Valores para los diversos parámetros utilizados por el algoritmo genético (tamaño de la población, probabilidad de cruza mutación, número máximo de generaciones, etc.)

En este capítulo hablaremos exclusivamente del primero de estos componentes: la representación usada por el algoritmo genético. La representación tradicional usada para codificar un conjunto de soluciones es el esquema binario en el cual un cromosoma es una cadena de la forma $\langle b_1, b_2, ..., b_m \rangle$ (ver figura 1.3), donde $b_1, b_2, ..., b_m$ se denominan alelos (ya sea ceros o unos).
Hay varias razones por las cuales suele usarse la codificación binaria en los AGs, aunque la mayoría de ellas se remontan al trabajo pionero de Holland en el área. En su libro, Holland dió una justificación teórica para usar codificaciones binarias. Holland comparó dos representaciones diferentes que tuvieran aproximadamente la misma capacidad de acarreo de información, pero de entre ellas, una tenía pocos alelos y cadenas largas (por ejemplo, cadenas binarias de 80 bits de longitud) y la otra tenía un número elevado de alelos y cadenas cortas (por ejemplo, cadenas decimales de longitud 24). Nótese que 2^{80} (codificación binaria) $\approx 10^{24}$ (codificación decimal). Holland argumentó que la primera codificación da pie a un grado más elevado de 'paralelismo implícito' porque permite más "esquemas "que la segunda (11^{24} contra 3^{80}).
El número de esquemas de una cadena se calcula usando $(c+1)^l$, donde c es la cardinalidad del alfabeto y l es la longitud de la cadena. Un "esquema"es una plantilla que describe un subconjunto de cadenas que comparten ciertas similitudes en algunas posiciones a lo largo de su longitud.

El hecho de contar con más esquemas favorece la diversidad e incrementa la probabilidad de que se formen buenos "bloques constructores "(es decir, la porción de un cromosoma que le produce una aptitud elevada a la cadena en la cual está presente) en cada generación, lo que en consecuencia mejora el desempeño del AG con el paso del tiempo de acuerdo al teorema de los esquemas . El "paralelismo implícito "de los AGs, demostrado por Holland, se refiere al hecho de que mientras el AG calcula las aptitudes de los individuos en una población, estima de forma implícita las aptitudes promedio de un número mucho más alto de cadenas cromosómicas a través del cálculo de las aptitudes promedio observadas en los "bloques constructores "que se detectan en la población. Por lo tanto, de acuerdo a Holland, es preferible tener muchos genes con pocos alelos posibles que contar con pocos genes con muchos alelos posibles. Esto es sugerido no sólo por razones teóricas (de acuerdo al teorema de los esquemas formulado por Holland),

sino que también tiene una justificación biológica, ya que en genética es más usual tener cromosomas con muchas posiciones y pocos alelos por posición que pocas posiciones y muchos alelos por posición.

Sin embargo, Holland también demostró que el paralelismo implícito de los AGs no impide usar alfabetos de mayor cardinalidad, aunque debe estarse siempre consciente de que el alfabeto binario es el que ofrece el mayor número de esquemas posibles por bit de información si se compara con cualquier otra codificación posible. No obstante, ha habido un largo debate en torno a cuestiones relacionadas con estos alfabetos no binarios, principalmente por parte de los especialistas en aplicaciones de los AGs. Como veremos en este capítulo, el uso de la representación binaria tiene varias desventajas cuando el AG se usa para resolver ciertos problemas del mundo real. Por ejemplo, si tratamos de optimizar una función con alta dimensionalidad (digamos, con 50 variables), y queremos trabajar con una buena precisión (por ejemplo, cinco decimales), entonces el mapeo de números reales a binarios generará cadenas extremadamente largas (del orden de 1000 bits en este caso), y el AG tendrá muchos problemas para producir resultados aceptables en la mayor parte de los casos, a menos que usemos procedimientos y operadores especialmente diseñados para el problema en cuestión. Ronald [18] resume las principales razones por las que una codificación binaria puede no resultar adecuada en un problema dado:

- Epístasis : el valor de un bit puede suprimir las contribuciones de aptitud de otros bits en el genotipo.
- Representación natural : algunos problemas (como el del viajero) se prestan de manera natural para la utilización de representaciones de mayor cardinalidad que la binaria (por ejemplo, el problema del viajero se presta de manera natural para el uso de permutaciones de enteros decimales).
- Soluciones ilegales : los operadores genéticos utilizados pueden producir con frecuencia (e incluso todo el tiempo) soluciones ilegales si se usa una representación binaria. En el resto de este capítulo discutiremos algunos esquemas de representación alternativos que han sido propuestos recientemente para lidiar con éstas y otras limitaciones de la representación binaria.

1.3. Técnicas de selección

La selección es el mecanismo por el cual soluciones más próximas al óptimo (individuos mejor adaptados) tienen mayor probabilidad de sobrevivir y ser elegidos (seleccionados) para

reproducirse. Supongamos que, en una determinada iteración del algoritmo, tenemos un conjunto de soluciones, que denominaremos población:

y que cada una de ellas (que denominamos individuos) tiene un valor de la función objetivo La selección supone que cuanto mayor sea $f(x_i)$, mayor sea la probabilidad p_i de que x_i sea elegido y se reproduzca. Una manera sencilla de hacer esto es tomar:

Debemos de garantizar que los mejores individuos tienen una mayor posibilidad de ser padres (reproducirse) frente a los individuos menos buenos. Debemos de ser cuidadosos para dar una oportunidad de reproducirse a los individuos menos buenos. Éstos pueden incluir material genético útil en el proceso de reproducción. Esta idea nos define la presión selectiva que determina en qué grado la reproducción está dirigida por los mejores individuos. Existen distintos métodos para la selección de los padres que serán cruzados para la creación de nuevos individuos, uno de los más utilizados se denomina función de selección proporcional a la función de evaluación.

Las técnicas de selección usadas en algoritmos genéticos pueden clasificarse en tres grandes grupos:

- Selección proporcional
- Selección mediante torneo
- Selección de estado uniforme

1.3.1. Selección Proporcional

Este nombre describe a un grupo de esquemas de selección originalmente propuestos por Holland en los cuales se eligen individuos de acuerdo a su contribución de aptitud con respecto al total de la población. Se suelen considerar 4 grandes grupos dentro de las técnicas de selección proporcional:

1. La Ruleta
2. Sobrante Estocástico
3. Universal Estocástica
4. Muestreo Determinístico

Adicionalmente, las técnicas de selección proporcional pueden tener los siguientes aditamentos:

1. Escalamiento Sigma
2. Jerarquías
3. Selección de Boltzmann

1.3.2. La Ruleta

Esta técnica fue propuesta por DeJong, y ha sido el método más comúnmente usado desde los orígenes de los algoritmos genéticos. El algoritmo es simple, pero ineficiente (su complejidad es $O(n^2)$. Asimismo, presenta el problema de que el individuo menos apto puede ser seleccionado más de una vez. Sin embargo, buena parte de su popularidad se debe no sólo a su simplicidad, sino al hecho de que su implementación se incluye en el libro clásico sobre AGs de David Goldberg. El algoritmo de la Ruleta (de acuerdo a DeJong) es el siguiente:

- Calcular la suma de valores esperados T
- Repetir N veces (N es el tamaño de la población):
  - Generar un número aleatorio entre 0.0 y T
  - Ciclar a través de los individuos de la población sumando los valores esperados hasta que la suma sea mayor o igual a r.
  - El individuo que haga que esta suma exceda el límite es el seleccionado.

1.3.3. Sobrante Estocástico

Propuesta por Booker y Brindle como una alternativa para aproximarse más a los valores esperados (*valesp*) de los individuos:

$$valesp_i = \frac{f_i}{\bar{f}}$$

La idea principal es asignar determinísticamente las partes enteras de los valores esperados para cada individuo y luego usar otro esquema (proporcional) para la parte fraccionaria.
El sobrante estocástico reduce los problemas de la ruleta, pero puede causar convergencia prematura al introducir una mayor presión de selección. El algoritmo es el siguiente:

1. Asignar de manera determinística el conteo de valores esperados a cada individuo (valores enteros).
2. Los valores restantes (sobrantes del redondeo) se usan probabilísticamente para rellenar la población. Hay 2 variantes principales:
   - **Sin reemplazo:** Cada sobrante se usa para sesgar el tiro de una moneda que determina si una cadena se selecciona de nuevo o no.
   - **Con reemplazo:** Los sobrantes se usan para dimensionar los segmentos de una ruleta y se usa esta técnica de manera tradicional.

1.3.4. Universal Estocástica

Propuesta por Baker con el objetivo de minimizar la mala distribución de los individuos en la población en función de sus valores esperados.
El método Universal Estocástica es análogo a una ruleta con M punteros igualmente espaciados entre si, de modo que con un solo lanzamiento se obtienen M ganadores. Este método no tiene sesgo de dispersión es la mínima posible. El algoritmo es como sigue:

Sum = 0

$Ptr = rand() \in [0, 1]$

$For\ i = 1\ to\ M$

Sum = $sum + N[i]$

$While\ (sum > ptr)\ do$

$Selectind[i]$

$Ptr = ptr + 1$

end

end

Donde $N[i]$ es el número esperado de copias para el individuo i-esimo. Debe tomarse en cuenta que SUS puede solo reducir el error de muestreo pero no eliminarlo completamente. Por otra parte la determinación del valor espado es muy sensible a la presencia de un individuo super adaptado en la población actual, pudiendo este llevarse un numeró elevado de copias generación tras generación y causar la convergencia prematura del algoritmo a un optimo local, especialmente para poblaciones pequeñas. Uniforma de resolver la convergencia prematura por presencia súper individuos es usar selección por ranking.

1.3.5. Muestreo Determinístico

Es una variante de la selección proporcional con la que experimentó DeJong.
Es similar al sobrante estocástico, pero requiere un algoritmo de ordenación.
El algoritmo del muestreo determinístico es el siguiente:

- Calcular $P_{select} = \frac{f_i}{\sum f_i}$
- Calcular $Valesp_i = P_{select} * n$($n$ =tamaño de la población)
- Asignar determinísticamente la parte entera de $Valesp_i$.

- Ordenar la población de acuerdo a las partes decimales (de mayor a menor).
- Obtener los padres faltantes de la parte superior de la lista.

1.3.6. Escalamiento Sigma

Es una técnica ideada para mapear la aptitud original de un individuo con su valor esperado de manera que el AG sea menos susceptible a la convergencia prematura. La idea principal de esta técnica es mantener más o menos constante la presión de selección a lo largo del proceso evolutivo. Usando esta técnica, el valor esperado de un individuo está en función de su aptitud, la media de la población y la desviación estándar de la población:

$$Valesp(i,t) = \begin{cases} 1 + \frac{f_i - \overline{f}}{2\sigma(t)}, & \text{si } \sigma(t) \neq 0; \\ 1, & \text{si } \sigma(t) = 0. \end{cases}$$

$$\sigma(t) = \sqrt{\frac{n\sum f(t)^2 - (\sum f(t))^2}{n^2}}$$

1.3.7. Selección por Jerarquías

Propuesta por Baker para evitar la convergencia prematura en las técnicas de selección proporcional. El objetivo de esta técnica es disminuir la presión de selección. En este caso, discutiremos el uso de jerarquías lineales, pero es posible también usar jerarquías no lineales, aunque la presión de selección sufre cambios más abruptos al usarse esta última.
Los individuos se clasifican con base en su aptitud, y se les selecciona con base en su rango (o jerarquía) y no con base en su aptitud. El uso de jerarquías hace que no se requiera escalar la aptitud, puesto que las diferencias entre las aptitudes absolutas se diluyen. Asimismo, las jerarquías previenen la convergencia prematura (de hecho, lo que hacen, es alentar la velocidad convergencia del algoritmo genético). El algoritmo de las jerarquías lineales es el siguiente:

- Ordenar (o jerarquizar) la población con base en su aptitud, de 1 a N (donde 1 representa al menos apto).
- Elegir $Max(1 \leq Max \leq 2)$
- Calcular $Min = 2 - Max$
- El valor esperado de cada individuo será: $valesp(i,t) = Min + (Max - Min)^{\frac{jerarquia(i,t)-1}{N-1}}$
  Baker recomendó $Max = 1{,}1$
- Usar selección proporcional aplicando los valores esperados obtenidos de la expresión anterior.

1.3.8. Selección de Boltzmann

Esta técnica fue propuesta por Goldberg y está basada en el recocido simulado: la idea es usar una función de variación de "temperatura"que controle la presión de selección. Se usa un valor alto de temperatura al principio, lo cual hace que la presión de selección sea baja. Con el paso de las generaciones, la temperatura disminuye, lo que aumenta la presión de selección. De esta manera se incita a un comportamiento exploratorio en las primeras generaciones y se acota a uno más exploratorio hacia el final del proceso evolutivo. El algoritmo es el siguiente: Típicamente, se usa la siguiente expresión para calcular el valor esperado de un individuo.

$$Valesp(i,t) = \frac{e^{\frac{f_i}{T}}}{\langle e^{\frac{f_i}{T}} \rangle_t} \tag{6.3}$$

donde T es la temperatura y $\langle\rangle_t$ denota el promedio de la población en la generación t.

1.3.9. Selección Mediante Torneo

Los métodos de selección proporcional antes descritos requieren de dos pasos a través de toda la población en cada generación:

1. Calcular la aptitud media (y, si se usa escalamiento sigma, la desviación estándar).
2. Calcular el valor esperado de cada individuo.

El uso de jerarquías requiere que se ordene toda la población (una operación cuyo costo puede volverse significativo en poblaciones grandes). La selección mediante torneo es similar a la de jerarquías en términos de la presión de selección, pero es computacionalmente más adecuada para implementarse en paralelo. Esta técnica fue propuesta por Wetzel y estudiada en la tesis doctoral de Brindle. La idea básica del método es seleccionar con base en comparaciones directas de los individuos. Hay 2 versiones de la selección mediante torneo:

- Determinística
- Probabilística

El algoritmo de la versión determinística es el siguiente:

- Barajar los individuos de la población.
- Escoger un número p de individuos (típicamente 2).
- Compararlos con base en su aptitud.

- El ganador del "torneo"es el individuo más apto.
- Debe barajarse la población un total de p veces para seleccionar N padres (donde N es el tamaño de la población).

El algoritmo de la versión probabilística es idéntico al anterior, excepto por el paso en que se escoge al ganador. En vez de seleccionar siempre al individuo con aptitud más alta, se aplica $flip(p)$[1] y si el resultado es cierto, se selecciona al más apto. De lo contrario, se selecciona al menos apto. El valor de p permanece fijo a lo largo de todo el proceso evolutivo y se escoge en el siguiente rango: $0{,}5 \leq p < 1$

1.3.10. Selección de Estado Uniforme

Esta técnica fue propuesta por Whitley y se usa en AGs no generacionales, en los cuales sólo unos cuantos individuos son reemplazados en cada generación (los menos aptos). Esta técnica suele usarse cuando se evolucionan sistemas basados en reglas (p.ej., sistemas clasificadores) en los que el aprendizaje es incremental.

En general, la técnica resulta útil cuando los miembros de la población resuelven colectivamente (y no de manera individual) un problema. Asimismo, los AGs generacionales se usan cuando es importante "recordar"lo que se ha aprendido antes. El algoritmo de la selección de estado uniforme es el siguiente:

- Llamaremos G a la población original de un AG.
- Seleccionar R individuos ($1 \leq R < M$) de entre los más aptos. Por ejemplo, $R = 2$.
- Efectuar cruza y mutación a los R individuos seleccionados. Llamaremos H a los hijos.
- Elegir al mejor individuo en H . (o a los μ mejores).
- Reemplazar los μ peores individuos de G por los μ mejores individuos de H.

1.4. Operadores evolutivos o de reproducción

Los individuos en las poblaciones intercambian información por medio de operadores evolutivos. Las siguientes secciones explican en forma breve los distintos tipos de operadores evolutivos.

1.4.1. Técnicas de Cruza

En los sistemas biológicos, la cruza es un proceso complejo que ocurre entre parejas de cromosomas. Estos cromosomas se alinean, luego se fraccionan en ciertas partes y posteriormente intercambian fragmentos entre sí. En computación evolutiva se simula la cruza intercambiando segmentos de cadenas lineales de longitud fija (los cromosomas). Aunque las técnicas de cruza básicas suelen aplicarse a la representación binaria, éstas son generalizables a alfabetos de cardinalidad mayor, si bien en algunos casos requieren de ciertas modificaciones. Comenzaremos por revisar las tres técnicas básicas de cruza:

- Cruza de un punto
- Cruza de dos puntos
- Cruza uniforme

Cruza de un punto

Esta técnica fue propuesta por Holland , y fue muy popular durante muchos años. Hoy en día, sin embargo, no suele usarse mucho en la práctica debido a sus inconvenientes. Puede demostrarse, por ejemplo, que hay varios esquemas que no pueden formarse bajo esta técnica de cruza.
Considere, por ejemplo, los esquemas siguientes:

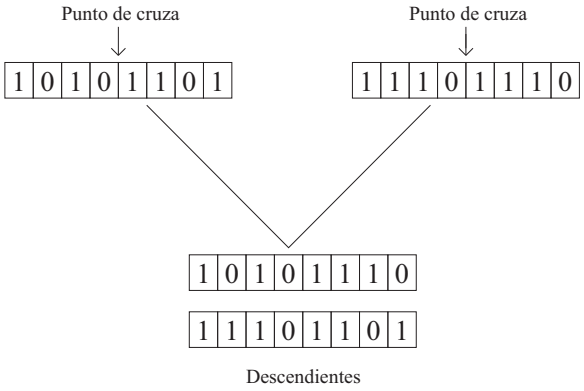


Figura 1.4: Cruza de un punto

Cruza de dos puntos

DeJong fue el primero en implementar una cruza de n puntos, como una generalización de la cruza de un punto. El valor $n = 2$ es el que minimiza los efectos disruptivos (o destructivos) de la cruza y de ahí que sea usado con gran frecuencia. No existe consenso en torno al uso de

valores para n que sean mayores o iguales a 3.
Los estudios empíricos al respecto proporcionan resultados que no resultan concluyentes respecto a las ventajas o desventajas de usar dichos valores. En general, sin embargo, es aceptado que la cruza de dos puntos es mejor que la cruza de un punto. Asimismo, el incrementar el valor de n se asocia con un mayor efecto disruptivo de la cruza.

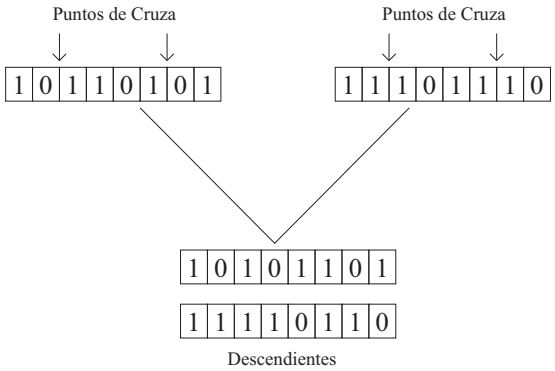


Figura 1.5: Cruza de dos puntos

Cruza uniforme

Esta técnica fue propuesta originalmente por Ackley, aunque se le suele atribuir a Syswerda. En este caso, se trata de una cruza de n puntos, pero en la cual el número de puntos de cruza no se fija previamente. La cruza uniforme tiene un mayor efecto disruptivo que cualquiera de las 2 cruzas anteriores. A fin de evitar un efecto excesivamente disruptivo, suele usarse con Pc =0.5. Algunos investigadores, sin embargo, sugieren usar valores más pequeños de Pc. Cuando se usa Pc =0.5 , hay una alta probabilidad de que todo tipo de cadena binaria de longitud L sea generada como máscara de copiado de bits.

1.4.2. Cruza para Representación Real

Si elegimos el uso directo de vectores de números reales para nuestra codificación, es entonces deseable definir operadores de cruza más acordes a esta representación . El énfasis principal es la capacidad de poder romperün cierto valor real, de manera análoga a como la cruza ordinaria rompe"segmentos de cromosoma al usarse representación binaria.

En el caso de los Algoritmos Genéticos con Codificación Real (AGCRs), este operador influ-

ye decisivamente sobre el nivel de diversidad en la población, y por ello, es un factor determinante para evitar el problema de la convergencia prematura.

Normalmente, el operador de cruce se aplica sobre parejas de padres, generando dos hijos para cada una de ellas, los cuales se introducen en la población. Sin embargo, existen operadores de cruce con múltiples padres, que combinan las características de más de dos padres para generar hijos. Su objetivo es introducir diversidad en la población, ya que mezclan información procedente de diversos padres. También se han presentado operadores de cruce con múltiples descendientes, que generan más de dos descendientes por cada grupo de padres. En este caso, un mecanismo de selección de descendientes limita el número de hijos que entran a formar parte de la nueva población.

Operadores de cruce basados en entornos (OCE): Estos operadores generan los genes de los descendientes a partir de valores tomados del entorno asociado a los padres, normalmente a partir de un intervalo cuyos extremos dependen de los valores de los genes de los cromosomas padre. Este tipo de operadores tienen carácter aleatorio. Los operadores propios de este grupo, BLX-α, BLX-α-β, SBX-η y $FR-d$. En la Figura 4 se observa el efecto de estos operadores.

Aleatoriamente se seleccionan dos padres:

$$P_1 = (c_1^1, \ldots, c_i^1, \ldots, c_n^1)$$
$$P_2 = (c_1^2, \ldots, c_i^2, \ldots, c_n^2)$$

BLX-α

. Se generan dos hijos:

$$H_k = (h_1^k, \ldots, h_i^k, \ldots, h_n^k),\ k = 1, 2$$

,donde h_i^k es un número elegido aleatoriamente en el intervalo $[C_{min} - \alpha.I, C_{max} - \alpha.I]$ con una distribución uniforme, donde

$$C_{max} = max\{c_i^1, c_i^2\}$$
$$C_{min} = min\{c_i^1, c_i^2\}$$
$$I = C_{max} - C_{min}$$

Cruce por simulación binaria (SBX-η):

Se generan dos hijos,

$$H_k = (h_1^k, ..., h_i^k, ..., h_n^k), k = 1, 2\ ,$$

donde: $h_i^1 = (1/2).[(1-\beta_k).c_i^1 + (1+\beta_k).c_i^2]$ y $h_i^2 = (1/2).[(1+\beta_k).c_i^1 + (1-\beta_k).c_i^2]$
$(\beta_k > 0)$ es una muestra de generador de números aleatorios con densidad:

$$p(\beta) = \begin{cases} \frac{1}{2}(\eta+1)\beta^{\eta}, & \text{si } 0 \leq \beta \leq 1 \\ \frac{1}{2}(\eta+1)\frac{1}{\beta^{\eta+2}}, & \text{si } \beta > 1 \end{cases}$$

Esta distribución puede obtenerse facilmente a partir de una uniforme $u(0,1)$ con una semilla aleatoria usando la transformación:

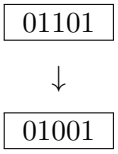
$$\beta(u) = \begin{cases} (2u)^{\frac{1}{\eta+1}}, & \text{si } u(0,1) \leq \frac{1}{2} \\ [2(1-u)]^{-\frac{1}{\eta+1}}, & \text{si } u(0,1) > \frac{1}{2} \end{cases}$$

1.4.3. Mutación

La mutación se considera como un operador secundario en los algoritmos genéticos canónicos. Es decir, su uso es menos frecuente que el de la cruza. En la práctica, se suelen recomendar porcentajes de mutación de entre 0.001 y 0.01 para la representación binaria.
Algunos investigadores, sin embargo, han sugerido que el usar porcentajes altos de mutación al inicio de la búsqueda, y luego decrementarlos exponencialmente, favorece el desempeño de un Algoritmo Genético.
Otros autores sugieren que $Pm = \frac{1}{L}$ (donde L es la longitud de la cadena cromosómica) es un límite inferior para el porcentaje óptimo de mutación.

La mutación es utilizada como un operador cuyo propósito es la exploración aleatoria de nuevas porciones de espacios de busqueda. Este operador es el encargado de introducir nuevo material genético en la búsqueda de soluciones, ya que el cruce no introduce ningún material nuevo.

Cuando la codificación utilizada es binaria el operador de mutación simplemente modifica el valor de un bit en la cadena con una probabilidad dada.



En el caso de los Algoritmos geneticos con representacion real entre los diferentes parametros para la mutacion encontramos Mutacion aleatoria, Mutacion No-uniforme y la mutacion

gaussiana.

Entre los operadores de mutación mas usados en la representación real se encuentran:

a) Mutación polinomial.

b) Mutación Uniforme.

c) Mutación Gaussiana.

a) **Mutación polinomial**

En la representación real uno de los métodos más conocidos es la mutación polinomial, en el cual se usa una función de densidad de probabilidad:

$$p(\delta) = 0{,}5(\eta_m + 1)(1 - |\delta|)^{\eta_m} \tag{1.1}$$

El parámetro η_m determina que tan lejos o cerca se encuentra el individuo con el gen mutado del padre. Para valores muy pequeños del η_m, la distancia es grande y además en algunas ocasiones puede incurrir en problemas de infactibilidad en cuanto al descendiente, sin embargo, a medida que este valor crece la distancia entre padre y descendiente disminuye y preserva la factibilidad.

Similar al SBX se genera un número aleatorio y posteriormente a partir de la distribución de probabilidad se encuentra δ_i a fin de que el área bajo la curva de probabilidad en el intervalo $[0, \delta_i]$ sea igual al número aleatorio r_i. Así:

$$\delta_i = \begin{cases} (2r_i)^{\frac{1}{\eta_m}+1} - 1 & r_i < 0{,}5 \\ 1 - [2(1 - r_i)]^{\frac{1}{\eta_m+1}} & r_i \geq 0{,}5 \end{cases} \tag{1.2}$$

El cálculo de la posición mutada será:

$$y_i^{1,1+t} = x_i^{1,t+1} + (x_i^U - x_i^L)\delta_i \tag{1.3}$$

Los valores de las cotas mínimas y máximas de x_i están representadas por x_i^L, x_i^U.

Se puede describir el algoritmo en forma general como:

- Seleccionar un padre.
- Generar un número aleatorio entre 0 y 1.
- Generar el valor de la posición mutada usando una función de densidad de probabilidad.

En este trabajo, los parámetros relacionados del cruce η_c y la mutación η_m, se les asignará el valor 20 recomendado por Deb.

b) **Mutación Uniforme**

La más simple mutación sería la creación de una solución al azar de todo el espacio de búsqueda:

$$(y_i^{(1,t+1)}) = r_i(x_i^{(U)} - x_i^{(L)})$$

donde r_i es un número aleatorio en $[0, 1]$. Este operador es independiente de la solución de los padres. En lugar de crear una solución desde el espacio de búsqueda completo, una solución en la vecindad de la solución de los padres con una distribución de probabilidad uniforme también se puede elegir:

$$y_i^{(1,t+1)} = x_i^{(1,t)} + (r_i - 0,5)\Delta_i$$

donde Δ_i es la perturbación máxima definida por el usuario como establecía la i-esima variable de decision. se debe tener cuidado para comprobar si el cálculo anterior $y_i^{(1,t+1)}$ cae fuera de los límites inferior y superior especificados.

c) **Mutación Gaussiana**

Dado un padre y un gen seleccionado para la mutación se genera un descendiente como:

$$y_i^{(1,t+1)} = x_i^{(1,t+1)} + N(0, \sigma_i)$$

donde $N(0, \sigma_i)$ es una distribución normal con media cero y desviación estandar σ_i, esta desviación estandar es un parámetro fijo y definido por el usuario. Este parámetro es importante y debe ser ajustado correctamente en un problema. Este parámetro también puede ser cambiado de forma adaptativa en cada generación por una regla predefinida.

Ajuste de parámetros

Los parámetros principales de un algoritmo genético son:

1. Tamaño de la población.
2. Porcentaje de cruza.
3. Porcentaje de mutación.

La forma óptima de definir estos parámetros en un algoritmo genético ha sido motivo de investigaciones desde los origenes mismos de la técnica, y no existe hasta la fecha una solución satisfactoria a este problema.

El tamaño de la población es un parámetro crucial para una aplicación exitosa de los algoritmos genéticos.

Experimentalmente Deb ([14]) observó que al maximizar una función bimodal:

$$f(x) = c_1 N(x, a_1, b_1) + c_2 N(x, a_2, b_2)$$

Donde $N(x, a_i, b_i)$ es la función gaussiana con media a_i y desviación estandar b_i, un tamaño de población pequeña hace que el algoritmo converja al máximo local, pero a medida que aumenta el número de individuos este converge al máximo global. Mas aún, el tamaño de la población está relacionado con la complejidad del problema, sin embargo, una población muy grande requiere muchos recursos computacionales. Aunque normalmente se elige un tamaño fijo de la población, también se puede considerar un tamaño de población variable. En este trabajo, se usará una población de 100 individuos, parámetro recomendado por Barán ([8]). De esta misma forma se debe establecer un número óptimo de generaciones (iteraciones) para asegurar la convergencia cercana a la solución óptima; Deb ([14]) recomienda un número de generaciones entre 250 y 1000, puesto que los algoritmos evolutivos han demostrado tener un buen desempeño en estos casos. Además, se establecerá un número de generaciones entre 1000 y 3000, sugeridos por Barán ([8]), debido a la alta dimensionalidad del problema.

Los parámetros sugeridos según DeJong, Shaffer y Goldberg ([12]) para cruce y mutación son:

- porcentaje de cruza de $0{,}65 - 0{,}95$
- porcentaje de mutación de $0{,}001 - 0{,}005$.

Sin embargo, Deb ([14]) recomienda un porcentaje de cruza de 0,9 y de mutación $1/n$ o $1/l$, donde n es el número de variables de decisión para la representación real y l es la longitud del string para la representación binaria. Estos parámetros permiten al algoritmo genético tener un desempeño razonablemente bueno.

Capítulo 2

Problema Optimización Valor de Orden (OVO)

En este capítulo presentaremos algunas definiciones y teoremas, referentes al problema OVO que pueden ser encontradas en [2], [3], [4] y [5], estas serán usadas para la comprensión y de desarrollo del presente trabajo.

Comenzaremos una pequeña introducción del problema OVO:

Dadas m funciones de valores reales f_1,...,f_m definidas en un dominio $\Omega \subset \mathbb{R}$ y un entero p $\in \{1, ..., m\}$. Para cada $x \in \Omega$ los valores de $f_1(x), ..., f_m(x)$ son ordenados tal como:

$$f_{i_1(x)}(x) \leq f_{i_2(x)}(x) \leq ... \leq f_{i_m(x)}(x)$$

La función de p orden $f : \Omega \to \mathbb{R}$ esta definida por

$$f(x) = f_{i_p(x)}(x)$$

El problema consiste en la minimización de la función p orden:

$$\begin{aligned} &\text{Minimizar } f(x) \\ &\text{s.a } x \in \Omega \end{aligned} \tag{2.1}$$

Este problema es denominado por sus primeros proponentes como problema OVO por su nombre en inglés (Order Value Optimization) e involucra la minimizacion de una funcion que en general es continua pero no es suave aún en el caso en que las funciones involucradas sean diferenciables. En el caso p=m estamos en presencia del clásico problema min-max.

2.1. Generalidades

Suponga que $\Omega \subset \mathbb{R}^n$ es cerrado y convexo, y $f_1, ..., f_m$ tienen derivadas parciales continuas en un conjunto abierto que las contiene. (El hecho de que sea cerrado y convexo se utilizará en

la condición de optimalidad, en la definición de los subproblemas del algoritmo principal y en las demostraciones de los Teoremas 2.2 y 2.3 a continuación.) Denotamos $g_j = \nabla f_j$ de ahora en adelante. Para todo $x, y \in \Omega$, $J = 1, .., m$, asumamos que

$$\|g_j(x)\|_\infty \leq c$$

y

$$\|g_j(y) - g_j(x)\|_\infty \leq L\|y - x\|\infty$$

En consecuencia, para todos $x, y \in \Omega$, $J = 1, ..., m$,

$$\|f_j(y) - f_j(x)\|_\infty \leq c\|y - x\|\infty \tag{2.2}$$

y

$$f_j(y) \leq f_j(x) + g_j(x)^T(y - x) + \frac{L}{2}\|y - x\|_\infty^2 \tag{2.3}$$

Dado $\varepsilon \geq 0$, $x \in \Omega$, Definiremos

$$I_\varepsilon(x) = \{j \in \{1, ..., m\} | f(x) - \varepsilon \leq f_j(x) \leq f(x) + \varepsilon\} \tag{2.4}$$

El teorema siguiente muestra la continuidad de la OVO-función.

Teorema 2.1. La función p-orden-valor f es continua.

Prueba. Suponga que $x^k \to x$, y supongamos que, para todos los k en algunas subsucesiones,

$$|f(x^k) - f(x)| \geq \delta > 0 \tag{2.5}$$

Por esa subsusecion, existe un índice $j \in \{1, ..., m\}$ de tal manera que

$$f(x^k) = f_j(x^k)$$

infinitamente muchas veces. Por lo tanto,

$$f_j(x^k) \geq f_\ell(x^k) \tag{2.6}$$

para al menos p índices $\ell \in \{1, ...m\}$. Por otra parte,

$$f_j(x^k) \leq f_\ell(x^k) \tag{2.7}$$

para al menos $m - p + 1$ índices ℓ en el conjunto $\{1, ..., m\}$. Como el número desubconjuntos de $\{1, ..., m\}$ es finito, un conjunto de índices ℓ que verifican (2.5) es repetido infinitamente en muchas ocasiones, y lo mismo sucede con un conjunto de índices ℓ que verifican (2.7). Por lo tanto, tomando límites en (2.6) y (2.7), obtenemos que

$$f_j(x) \geq f_\ell(x)$$

para al menos p índices $\ell \in \{1, ..., m\}$ y

$$f_j(x) \leq f_\ell(x)$$

por lo menos $m - p + 1$ índices $\ell \in \{1, ..., m\}$.

Por lo tanto,

$$f(x) = f_j(x)$$

Pero $f_j(xk) \to f_j(x)$, por lo que esta en contradicción con (2.5).

El teorema siguiente proporciona una condición necesaria para la optimalidad del problema OVO.

Definición 2.1.1 *Diremos que x es ε-óptima si*

$$\mathrm{D} \equiv \{d \in \mathbb{R}^n \mid x + d \in \Omega \text{ y } g_j(x)^{\mathrm{T}} d < 0,\ \forall j \in I_\varepsilon(x)\} = \emptyset \tag{2.8}$$

Teorema 2.1.1 *Si $x_* \in \Omega$ es un minimizador local de $f(x)$ en $x \in \Omega$ y $\varepsilon > 0$, entonces x_* es ε-óptima.*

[Prueba.] Supongamos por reducción al absurdo que (2.8) no es cierto. Entonces, existen $d \in \mathbb{R}^n$ y $\overline{\alpha} > 0$ tal que $x_* + d \in \Omega$ y

$$f_j(x_* + \alpha d) < f_j(x_*), \qquad \forall \alpha \in (0, \overline{\alpha}] \tag{2.9}$$

para todo $j \in I_\varepsilon(x_*)$.
Así, para todo $j \in \{1, \ldots, m\}$ con $f_j(x_*) = f(x_*)$, se tiene que $j \in I_\varepsilon(x_*)$ y luego se obtiene, bajo el supuesto de la prueba, que (2.9) permanece.

Definamos

$$\varepsilon_1 = \min_{f_j(x_*) < f(x_*)} \{f(x_*) - f_j(x_*)\} \in (0, +\infty],$$

$$\varepsilon_2 = \min_{f_j(x_*) > f(x_*)} \{f_j(x_*) - f(x_*)\} \in (0, +\infty]$$

sea $\widetilde{\alpha} \leq \overline{\alpha}$ tal que

$$|f_j(x_* + \alpha d) - f_j(x_*)| < \frac{\min\{\varepsilon_1, \varepsilon_2\}}{2}$$

para todo $j \in \{1, \ldots, m\}$, $\alpha \in (0, \widetilde{\alpha}]$.

Por lo tanto, para todo $\alpha \in (0, \widetilde{\alpha}]$, el índice j tal que

$$f(x_* + \alpha d) = f_j(x_* + \alpha d)$$

es uno de los índices j tal que $f(x_*) = f_j(x_*)$. En otras palabras, este índice pertenece a $I_0(x_*) \subset I_\varepsilon(x_*)$.

Pero por (2.9),

$$f_j(x_* + \alpha d) < f_j(x_*)$$

para todo $\alpha \in (0, \widetilde{\alpha}]$. Por lo tanto, x_* no es minimizador local, esto es una contradicción con la hipotesis. □

A continuación se describe el algoritmo principal.

Algoritmo 2.1.2 *Sea $x_0 \in \Omega$ un punto inicial arbitrario. Sea $\theta \in (0, 1)$, $\Delta > 0$, $\varepsilon > 0$, $0 < \sigma_{\text{mín}} < \sigma_{\text{máx}} < 1$, $\eta \in (0, 1]$.*

Dado $x_k \in \Omega$, los pasos de la k-ésima iteración son:

Paso 1. *(Solución del subproblema)*

Definamos

$$M_k(d) = \max_{i \in I_\varepsilon(x_*)} g_i(x_k)^{\mathrm{T}} d \tag{2.10}$$

Considere el subproblema

$$\text{Minimizar } \; M_k(d) \; \text{ s.a. } \; x_k + d \in \Omega, \; ||d||_\infty \leq \Delta \tag{2.11}$$

Note que (2.11) es equivalente al problema de optimización convexa

$$\begin{aligned} &\text{Minimizar } \; w \\ &g_i(x_k)^{\mathrm{T}} \leq w, \quad \forall j \in I_\varepsilon(x_k) \\ &x_k + d \in \Omega, \; ||d||_\infty \leq \Delta \end{aligned}$$

Sea d_k una solución (más adelante veremos que no es necesario calcular la misma) tal que $x_k + d \in \Omega$, $||d_k||_\infty \leq \Delta$ y

$$M_k(d) \leq \eta M_k(\overline{d_k}) \tag{2.12}$$

Si $M_k(d) = 0$ parar.

Paso 2.

Sea $\alpha \leftarrow 1$. Si

$$f(x_k + \alpha d_k) \leq f(x_k) + \theta \alpha M_k(x_k) \tag{2.13}$$

fijar $\alpha_k = \alpha$, $x_{k+1} = x_k + \alpha_k d_k$ *y finaliza la iteración. De lo contrario, seleccione* $\alpha_{\text{nuevo}} \in [\sigma_{\text{mín}}\alpha, \sigma_{\text{máx}}\alpha]$, *fijar* $\alpha \leftarrow \alpha_{\text{nuevo}}$ *y repetir hasta que se cumpla (2.13).*

Lo que sigue es un lema técnico que será útil para la prueba de convergencia.

Lema 2.1.1 *Supongamos que* $a_1, a_2, \ldots, a_r$ *son números reales tales que*

$$a_1 \leq a_2 \leq \cdots \leq a_q \leq \cdots \leq a_r$$

Supongamos que $\beta > 0$ *y* $b_1, b_2, \ldots, b_r \in \mathbb{R}$ *tales que*

$$b_j \leq a_j - \beta, \qquad \forall j = 1, \cdots, r$$

y $$b_{i_1} \leq b_{i_2} \leq \cdots \leq b_{i_q} \leq \cdots \leq b_{i_r}$$

Entonces

$$b_{i_q} \leq a_{i_q} - \beta.$$

[**Prueba.**] Claramente

$$b_{i_q} \leq a_{i_q} - \beta, \qquad b_{i_q} \leq b_{i_{q+1}} \leq a_{i_{q+1}} - \beta, \ldots, b_{i_q} \leq b_{i_r} \leq a_{i_r} - \beta$$

Entonces, $\quad b_{i_q} \leq \text{mín}\{a_{i_q}, \ldots, a_{i_r}\} - \beta$

Pero como $a_1 \leq a_2 \leq \cdots \leq a_q \leq \cdots \leq a_r$, tenemos que

$$\text{mín}\{a_{i_q}, \ldots, a_{i_r}\} \leq a_q.$$

Por lo tanto,

$$b_{i_q} \leq a_q - \beta.$$

□

En el siguiente teorema, podemos afirmar que, si la iteración x_k no es ε-óptima, entonces x_{k+1} esta bien definida y α_k es acotado lejos de cero.

Teorema 2.1.3 *Suponga que* $x_k \in \Omega$ *es la* k*-ésima iteración del Algoritmo 2.1.2. Entonces:*

a) *El algoritmo para en* x_k *si y sólo si,* x_k *es* ε*-óptima.*

b) *Si el algoritmo no para en* x_k*, entonces la iteración esta bien definida y*

$$\alpha_k \geq \text{mín}\left\{\frac{2\sigma_{\text{mín}}\gamma_k(1-\theta)}{L\Delta^2}, \frac{\varepsilon\sigma_{\text{mín}}}{3c\Delta}\right\} \tag{2.14}$$

donde

$$\gamma_k = -\max_{j \in I_\varepsilon(x_k)}\{g_j(x_k)^{\mathrm{T}}\} > 0. \tag{2.15}$$

[Prueba.] Si el algoritmo para en x_k, entonces $M_k(d_k) = 0$. Por lo tanto, por (2.12), $M_k(\overline{d_k}) = 0$. Así, $M_k(d) \geq 0$ para todo $d \in D$ tal que $||d||_\infty \leq \Delta$. Por lo tanto, $M_k(d_k) \geq 0$ para todo $d \in D$. Esto implica que x_k es ε-óptima.

Reciprocamente, si x_k es ε-óptima, debemos tener que $M_k(\overline{d_k}) = 0$, así $M_k(d_k) = 0$ y el algoritmo para en x_k.
Si el algoritmo no para en x_k, entonces $M_k(\overline{d_k}) < 0$, por lo tanto,

$$-\gamma_k = \max_{j \in I_\varepsilon(x_k)} \{g_j(x_k)^{\mathrm{T}}\} < 0.$$

Suponga que

$$\alpha \in \left[0 \, , \frac{2\gamma_k(1-\theta)}{L\Delta^2}\right]$$

Entonces,

$$\frac{L\alpha\Delta^2}{2} \leq (1-\theta)\gamma_k$$

Así

$$\frac{L\alpha\Delta^2}{2} \leq (\theta - 1)g_j(x_k)^{\mathrm{T}}d_k, \quad \forall j \in I_\varepsilon(x_k)$$

Por lo tanto,

$$g_j(x_k)^{\mathrm{T}}d_k + \frac{L\alpha\Delta^2}{2} \leq \theta g_j(x_k)^{\mathrm{T}}d_k, \quad \forall j \in I_\varepsilon(x_k)$$

En consecuencia, puesto que $||\alpha d_k||_\infty \leq \Delta$,

$$\alpha g_j(x_k)^{\mathrm{T}}d_k + \frac{L\alpha^2||d_k||_\infty^2}{2} \leq \alpha\theta g_j(x_k)^{\mathrm{T}}d_k, \quad \forall j \in I_\varepsilon(x_k)$$

Así

$$f_j(x_k) + g_j(x_k)^{\mathrm{T}}(\alpha d_k) + \frac{L}{2}||\alpha d_k||_\infty^2 \leq f_j(x_k) + \alpha\theta g_j(x_k)^{\mathrm{T}}d_k, \quad \forall j \in I_\varepsilon(x_k)$$

entonces, por (2.3)

$$f_j(x_k + \alpha d_k) \leq f_j(x_k) + \alpha\theta g_j(x_k)^T d_k \quad \forall j \in I_\varepsilon(x_k)$$

Por lo tanto, hemos demostrado que, $\alpha \in [0, 2\gamma_k(1-\theta)/(L\Delta^2)]$,

$$f_j(x_k + \alpha d_k) \leq f_j(x_k) + \alpha\theta M_k(d_k) \; \forall j \in I_\varepsilon(x_k) \tag{2.16}$$

Por otro lado, si $\alpha \in [0, \varepsilon/(3c\Delta)]$, tenemos que $\alpha c\Delta \leq \varepsilon/3$ así, $\alpha c\|d_k\|_\infty \leq \varepsilon/3$, por lo tanto, $c\|\alpha d_k\|_\infty \leq \varepsilon/3$, entonces por (2.2)

$$|f_j(x_k + \alpha d_k) - f_j(x_k)| \leq \frac{\varepsilon}{3} \forall j = 1, ..., m \tag{2.17}$$

entonces, para todo $\ell = 1, ..., p$,

$$f_{i_\ell(x_k)}(x_k + \alpha d_k) \geq f_{i_\ell(x_k)}(x_k) + \tfrac{\varepsilon}{3} \geq f(x_k) + \tfrac{\varepsilon}{3}$$

Esto significa que al menos p elementos del conjunto

$$\{f_1(x_k + \alpha d_k), ..., f_m(x_k + \alpha d_k)\}$$

Son menores o iguales a $f(x_k) + \varepsilon/3$ y por lo menos $m - p + 1$ elementos de ese conjunto son mayores o iguales que $f(x_k) - \varepsilon/3$

Por lo tanto,

$$f_j(x_k + \alpha d_k) = f_{i_p(x_k + \alpha d_k)}(x_k + \alpha d_k) \in [f(x_k) - \frac{\varepsilon}{3}, f(x_k) + \frac{\varepsilon}{3}] \tag{2.18}$$

Supongamos que, $j \notin I_\varepsilon(x_k)$, así, o $f_j(x_k) < f(x_k) - \varepsilon$ o $f_j(x_k) > f(x_k) + \varepsilon$. En el primer caso, por (2.17), tenemos

$$f_j(x_k + \alpha d_k) < f(x_k) - \tfrac{2}{3}\varepsilon,$$

así, por (2.18),

$$f_j(x_k + \alpha d_k) < f(x_k + \alpha d_k)$$

Análogamente, si $f_j(x_k) > f(x_k) + \varepsilon$, a continuación,

$$f_j(x_k + \alpha d_k) > f(x_k + \alpha d_k)$$

Por lo tanto,

$$f(x_k + \alpha d_k) = f_j(x_k + \alpha d_k) \text{ para algunos } j \in I_\varepsilon(x_k)$$

Escribamos,

$$I_\varepsilon(x_k) = \{j_1, ..., j_\nu\} = \{j'_1, ..., j'_\nu\}$$

donde,

$$f_{j_1}(x_k) \leq ... \leq f_{j_\nu}(x_k)$$

y

$$f_{j'_1}(x_k + \alpha d_k) \leq ... \leq f_{j'_\nu}(x_k + \alpha d_k)$$

Es evidente que existe $q \in \{1, ..., \nu\}$ tales que $i_p(x_k) = j_q$

Ahora, los índices $j \notin I_\varepsilon(x_k)$ de tal manera que $f_j(x_k) < f(x_k)$ son los mismos que los índices $j \notin I_\varepsilon(x_k)$ de tal manera que $f_j(x_k + \alpha d_k) < f(x_k)$ y, por otra parte, los índices $j \notin I_\varepsilon(x_k)$ tal que $f_j(xk) > f(x_k)$ son los mismos que los índices $j \notin I_\varepsilon(x_k)$ de tal manera que $f_j(x_k + \alpha d_k) > f(x_k)$. Entonces,

$$i_p(x_k + \alpha d_k) = j'_q$$

Luego, por (2.16) y el lema 2.1, tenemos que, cuando

$$\alpha \in [0, min\{\frac{2\gamma_k(1-\theta)}{L\Delta^2}, \frac{\varepsilon}{3c\Delta}\}] \tag{2.19}$$

tenemos:

$$f_{j'_q}(x_k + \alpha d_k) \leq f_{j_q}(xk) + \alpha\theta M_k(d_K)$$

Por lo tanto,

$$f_{i_p(x_k+\alpha d_k)}(x_k + \alpha d_k) \leq f_{i_p(x_k)}(x_k) + \alpha\theta M_k(d_k)$$

Así,

$$f(x_k + \alpha d_k) \leq f(x_k) + \alpha\theta M_k(d_k)$$

Por lo tanto, siempre que sea (2.19) se lleva a cabo, la prueba (2.13) debe ser titular. Esto significa que un valor de α que no satisface (2.13) no puede ser menor que $min\{2\gamma_k(1-\theta)/(L\Delta^2), \varepsilon/(3c\Delta)\}$. Por tanto, la aceptación α Debe satisfacer:

$$\alpha_k \geq min\{\tfrac{2\sigma_{min}\gamma_k(1-\theta)}{L\Delta^2}, \tfrac{\varepsilon\sigma_{min}}{3c\Delta}\},$$

como queríamos demostrar.

□

El resultado principal de convergencia se da en el teorema 2.1.4. En primer lugar, tenemos que demostrar un lema de preparación simple.

Lema 2.1.2 *Si $\{x_k\}$ es una sucesión infinita generada por el Algoritmo 2.1.2, entonces*

$$\lim_{k\to\infty} f(x_k) = -\infty \tag{2.20}$$

o

$$\lim_{k\to\infty} M_k(d_k) = 0 \tag{2.21}$$

[**Prueba.**] Si (2.20) no se cumple, entonces por (2.13), tenemos que

$$\lim_{k\to\infty} \alpha_k M_k(d_k) = 0$$

Por el Teorema 2.1.3, implica que (2.21) se lleva a cabo. □

Teorema 2.1.4 *Supongamos que $x_* \in \Omega$ es un punto límite de la sucesión generada por el Algoritmo 2.1.2. Entonces x_* es ε-óptima.*

[**Prueba.**] Dado que $f(x_{k+1}) \leq f(x_k)$ para todo $k = 0, 1, \ldots$ y x_* es un punto límite de $\{x_k\}$ entonces $f(x_k) \to f(x_*)$. Por lo tanto, por el Lema 2.1.2, (2.21) se lleva a cabo. Sea K una sucesión infinita de índices tales que

$$\lim_{k\in K} x_k = x_*$$

Si x_* no es ε-óptima, entonces existe $\gamma > 0$ y $d \in \mathbb{R}^n$ tal que $x_* + d \in \Omega$ y

$$g_j(x_*)^{\mathrm{T}} d \leq -\gamma, \qquad \forall j \in I_\varepsilon(x_*) \tag{2.22}$$

Sin perdida de generalida podemos suponer que $\|d\| \leq \frac{\Delta}{2}$. Por continuidad, para k suficienttemente grande, $k \in K$, definiendo

$$\hat{d}_k = d + x_* - x_k,$$

Tenemos que $\|\hat{d}_k\|_\infty \leq \delta$, $x_k + \hat{d}_k \in \Omega$ y

$$\lim_{k\in K} \hat{d}_k = d \tag{2.23}$$

Por (2.21), tenemos que $\lim_{k\to\infty} M_k(\overline{d_k}) = 0$. Por lo tanto, $\lim \text{ínf}_{k\to\infty} M_k(\hat{d}_k) \geq 0$. para todo $k \in K$ existe $j \in I_\varepsilon(x_k)$ tal que $g_j(x_k)^T \hat{d}_k = M_k(\hat{d}_k)$. Dado que $I_\varepsilon(x_k)$ es finito, existe j tal que $g_j(x_k)^T \hat{d}_k = M_k(\hat{d}_k)$ infinitas veces. Por lo tanto, para j particular,

$$\lim_{k\in K} \text{ínf}\, g_j(x_k)^T \hat{d}_k = 0$$

Luego, tomando limites,

$$g_j(x_*)^T d = 0 \tag{2.24}$$

Pero, en infinitos índices, ya que $j \in I_\varepsilon(x_k)$,

$$f(x_k) - \varepsilon \leq f_j(x_k) \leq f(x_k) + \varepsilon,$$

Por lo tanto, tomando limites,

$$f(x_*) - \varepsilon \leq f_j(x_*) \leq f(x_*) + \varepsilon,$$

Así, $j \in I_\varepsilon(x_*)$.

Por lo tanto, (2.24) contradice (2.22). □

2.2. Cómputo de una solución aproximada del subproblema

Si n es grande $I_\varepsilon(x_k)$ contiene muchos elementos, calcular la solucion exacta de (2.10) puede ser muy costoso. En estos casos, una solución aproximada d_k que satisfaga (2.12) se puede calcular siguiendo el procedimiento descrito a continuación. Para simplificar la descripción se supone que $Int(\Omega)$, el interior del conjunto convexo Ω, no es vacío. Elegir $\eta' \in (\eta, 1)$. Las iteraciones sucesivas x_k pertenecerán a $Int(\Omega)$. Supongamos que existe un procedimiento que calcula una sucesión $\{s_\nu\}$, donde $s_\nu \in \mathbb{R}^n$, tal que $s_\nu \to \overline{d_k}$ y una sucesión de cotas c_ν tal que $c_\nu \to M_k(\overline{d_k})$ y

$$c_\nu \leq M_k(\overline{d_k})$$

para todo $\nu = 0, 1, 2, ...$ Defina, para todo $\nu = 0, 1, 2, ...,$

$$\lambda_\nu = min\{1, max\{\lambda \leq 0 | x_k + \lambda s_\nu \in \Omega, \|\lambda s_\nu\|_\infty \leq\}\}.$$

Dado que Ω es convexo y x_k esta en su interior tenemos que

$$\lambda_\nu \to 1$$

Así,

$$x_k + \lambda_\nu s_\nu \to x_k + \overline{d_k}$$

y

$$M_k(\lambda_\nu s_\nu) \to M_k(\overline{d_k}).$$

Por lo tanto, tomando ν suficientemente grande, obtenemos λ_ν tal que

$$M_k(\lambda_\nu s_\nu) \leq \eta' c_\nu \leq \eta' M_k(\overline{d_k}) \tag{2.25}$$

Así,

$$M_k(\tfrac{\eta}{\eta'} \lambda_\nu s_\nu) \leq \eta M_k(\overline{d_k})$$

Esto implica que, tomando

$$d_k = \frac{\eta}{\eta'}\lambda_\nu s_\nu,$$

La condición (2.12) es satisfecha, $x_k + \alpha d_k$ sigue estando en el interior para todo $\alpha \in [0, 1]$. Por tanto, x_{k+1} esta en el interior y el proceso se puede repetir en la siguiente iteración.

Capítulo 3

Implementación Numérica

En el presente capítulo implementaremos un algoritmo genético con selección elitista, en el cual usamos una codificacion o representacion real de la poblacion, además usamos operadores de reproducción, en el caso del cruzamiento: cruza BLX-α y en el caso de la mutacion: Mutacion Gaussiana. Ilustramos el comportamiento del algoritmo aplicado a un problema de ajuste de datos.

En primer lugar, mostramos a continuación el código en matlab del algoritmo usado.

3.1. Programa Principal

% Algoritmo Genetico Elitista para resolver el problema OVO

%==========================

clc;

clear;

lchrom = 20 % long. del cromosoma

popsize = 200*lchrom % tamaño de la poblacion

pcross = 0.4 % prob.de crossover

pmutation = 0.0001 % prob. de mutacion

%elitismo = 1 % 0 sin elitismo, 1 si

maxgen = 86 % maxima cantidad de generaciones

p=20; % el valor del orden

while (p $\leq$ 43),

%Construir el intervalo MinInterv=-10; MaxInterv=10; %Construir la poblacion inicial

Pop =pop2(MinInterv,MaxInterv,popsize);

```
z = Pop;
Fitness=orden(z,p);

    gen = 0;
while (gen ≤ maxgen),
gen = gen + 1;
[b,eli]=max(Fitness);
NewPop = generarealr2(Pop, Fitness, pcross, pmutation,MinInterv,MaxInterv,p);
NewPop(:,1)=Pop(:,eli);
w = NewPop;
Fitness =orden(w,p);
[FitMax,maxf] = max(Fitness);

    Pop = NewPop;
xval=NewPop(:,maxf);
end
xval.
```

Nota: El programa anterior le llamaremos AGRealOrdenp.

En el siguiente codigo (en matlab) es determinada la función orden p (p=1,2,...,46):

```
% Función de orden p

    function y = orden(x,p);
H=[f1(1,x);f1(2,x);f1(3,x);f1(4,x);f1(5,x);f1(6,x);f1(7,x);f1(8,x);f1(9,x);...
f1(10,x);f1(11,x);f1(12,x);f1(13,x);f1(14,x);f1(15,x);f1(16,x);f1(17,x);...
f1(18,x);f1(19,x);f1(20,x);f1(21,x);f1(22,x);f1(23,x);f1(24,x);f1(25,x);...
f1(26,x);f1(27,x);f1(28,x);f1(29,x);f1(30,x);f1(31,x);f1(32,x);f1(33,x);...
f1(34,x);f1(35,x);f1(36,x);f1(37,x);f1(38,x);f1(39,x);f1(40,x);f1(41,x);...
f1(42,x);f1(43,x);f1(44,x);f1(45,x);f1(46,x)];
% H es una matriz de valores de las m=46 funciones
J=sort(H,1,'ascend'); % J es una matriz de orden por columna de forma creciente bajo la instruccion sort de la matriz H
y=J(p,:); % y es una matriz fila con los valores de la fila p de la matriz J
y=-y;
```

Ejemplo:

Para ilustrar el comportamiento del algoritmo, consideramos un ejemplo de ajuste de datos el cual fue resuelto en la referencia [3] usando un método tipo Cauchy.

Supongamos que deseamos ajustar el modelo,

$$y(t,x) = x_1 + x_2 t + x_3 t^2 + x_4 t^3$$

para un conjunto de datos (t_i, y_i), $i = 1, ..., m$. La función a minimizar es obtenida del conjunto de funciones error

$$f_i(x) = (y(t_i, x) - y_i)^2$$

Teniendo en cuenta la solución,

$$x^* = (x_1^*, x_2^*, x_3^*, x_4^*) = (0, 2, -3, 1),$$

Generamos, aleatoriamente la data por,

$$\begin{gathered} w_i = y(t_i, x^*), \\ t_i = -1 + 0{,}5i \ i = 1, .., m, \ m = 46, \\ y_i = 10 \ i = 7, .., 16 \ \text{ y} \\ y_i = w_i + r_i, \ \text{ en otros casos,} \end{gathered}$$

donde r_i es aletorio entre -0.01 y 0.01. Por lo tanto, $y_7, ..., y_{16}$, son observaciones malas o "outliers "obtenidas por simulación. La data esta ubicada en la figura 3.1.

Por otra parte, las soluciones que obtuvimos usando el algoritmo evolutivo propuesto, estan dadas en la tabla 1 para distintos valores de p, donde p define la función de orden p, obteniendo para p=36 el suiguiente resultado:

$$\begin{gathered} \bar{x} = (\bar{x_1}, \bar{x_2}, \bar{x_3}, \bar{x_4}) = (0.0004, 1.9998, -3.0000, 1.0000), \\ f(\bar{x}) = 0.0402 \end{gathered}$$

i	t_i	y_i	i	t_i	y_i	i	t_i	y_i
1	-1.0000	-5.8000	17	0.6000	0.5360	33	2.2000	0.3280
2	-0.9000	-5.1590	18	0.7000	0.4730	34	2.3000	1.0970
3	-0.8000	-4.2320	19	0.8000	-0.0080	35	2.4000	1.1440
4	-0.7000	-3.4130	20	0.9000	0.2990	36	2.5000	1.6750
5	-0.6000	-2.6960	21	1.0000	0.2000	37	2.6000	2.2960
6	-0.5000	-1.6750	22	1.1000	-0.2990	38	2.7000	3.4130
7	-0.4000	10.0000	23	1.2000	0.0080	39	2.8000	4.2320
8	-0.3000	10.0000	24	1.3000	-0.0730	40	2.9000	5.1590
9	-0.2000	10.0000	25	1.4000	-0.5360	41	3.0000	6.2000
10	-0.1000	10.0000	26	1.5000	-0.5750	42	3.1000	6.9610
11	0	10.0000	27	1.6000	-0.5840	43	3.2000	8.6480
12	0.1000	10.0000	28	1.7000	-0.5570	44	3.3000	9.6670
13	0.2000	10.0000	29	1.8000	-0.4880	45	3.4000	11.2240
14	0.3000	10.0000	30	1.9000	0.0290	46	3.5000	12.9250
15	0.4000	10.0000	31	2.0000	-0.2000			
16	0.5000	10.0000	32	2.1000	0.0310			

Figura 3.1:

p	x_1	x_2	x_3	x_4	$fobj$
20	−0,1960	2,0016	−3,0027	1,0006	0,0000
21	0.2898	2.1330	-3.4764	1.1365	0.0137
22	0.2665	2.1107	-3.4446	1.1358	0.0187
23	0.2608	2.1527	-3.2985	1.0590	0.0351
24	0.2565	2.1771	-3.3712	1.0814	0.0262
25	0.2095	2.1742	-3.3330	1.0731	0.0289
26	0.1673	2.1149	-3.3061	1.0901	0.0326
27	0.1952	2.1474	-3.3540	1.1020	0.0351
28	-0.0031	2.0036	-3.0010	1.0000	0.0401
29	0.1958	1.6704	-2.8308	0.9729	0.0426
30	-0.0571	2.1659	-3.1272	1.0282	0.0439
31	0.0993	1.8389	-2.9201	0.9876	0.0413
32	-0.0224	2.0508	-3.0310	1.0054	0.0415
33	-0.0200	2.0361	-3.0176	1.0025	0.0413
34	0.0206	2.1133	-3.1126	1.0226	0.0629
35	0.0096	2.0153	-3.0202	1.0045	0.0446
36	0.0004	1.9998	-3.0000	1.0000	0.0402
37	8.5968	-8.1420	-0.9937	1.1061	9.3787
38	9.1908	-8.5327	-1.0137	1.1142	11.7662
39	9.9883	-9.3235	-0.8255	1.0960	14.4629
40	7.4122	-0.0932	-7.1706	2.2765	17.1600
41	6.1998	2.5339	-8.6074	2.5049	17.1637
42	6.2138	2.7182	-8.6283	2.4799	18.2686
43	5.8688	2.4626	-7.6065	2.1383	20.5587

Tabla1: Solucion del problema OVO mediante el algoritmo AGRealOrdenp

Observe que como generamos 10 “outliers ”, el valor correcto de p debe ser 36. De esta forma, si observamos la tabla anterior, notamos que los resultados fueron satisfactorios debido a que para p>36 la solucion obtenida esta lejos de x^* optimo y sus valores objetivos son elevados. Este ejemplo sugiere que en situaciones reales, cuando no se conoce el número de datos errados (outliers), podríamos probar con diferentes valores de p y observando los resultados nos permitirían tener una solución aceptable.

Conclusión

Se ha realizado un estudio de las caracteristicas y propiedades teóricas del problema de minimización de una función de orden p, además de los aspectos teóricos y computacionales de los algoritmos evolutivos, para abordar todo lo relacionado a su descripción y obtener un estudio general de los mismos. La aplicación de las técnicas de los algoritmos evolutivos han sido de gran utilidad para abordar aquellos problemas donde los métodos clásicos no resuelven de manera eficiente,el problema OVO no se escapa de esto, es decir a veces no son tan fáciles de resolver y los algoritmos evolutivos se comportan efizcamente para su resolución, una ventaja adicional es que la técnica para resolver el problema OVO, un algoritmo evolutivo permite obtener optimos globales lo cual no es así para la mayoria de algoritmos de optimizacion continua.

Así, se ha diseñado e implementado en matlab un algoritmo evolutivo elitista para resolver el problema OVO (Orden Value Optimization), donde se obtuvieron resultados satisfactorios, en el problema considerado, similares a los obtenidos en [3] usando un método tipo Cauchy.

Se espera que el presente trabajo sirva como referencia para otros investigadores, docentes y/o estudiantes interesados en la temática abordada.

Bibliografía

[1] Ackley David H. *A Connectionist Machine for Genetic Hillclimbing.* Kluwer Academic Publishers, Boston, Massachusetts, (1987).

[2] Andreani R., Dunder C. y Martínez J. M., *Nonlinear programming reformulation of the order-value optimization problem*, tecnical report, institute of mathematics, University of Campinas, Brasil, (2005).

[3] Andreani R., Dunder C. y Martíinez J. M., *Order-value optimization: formulation and solution by means of a primal Cauchy method*, mathematical methods of operation research **58**, pp. 387–399, (2003).

[4] Andreani R., Martínez J. M., Martínez L. y Yano F. *Low the order-value optimization and aplications*, tecnical report, institute of mathematics, University of Campinas, Brasil, (2007).

[5] Andreani R., Martínez J. M., Salvatierra M. y Yano F. *Quasi-Newton methods for order-value optimization and Value-at-Ristk*, pacific journal of optimization **2**, pp. 11–33 (2006).

[6] Baker J. E. *Adaptive Selection Methods for Genetic Algorithms.* In John J. Grefenstette, editor, Proceedings of the First International Conference on Genetic Algorithms, pp. 101–111. Lawrence Erlbaum Associates, Hillsdale, New Jersey, (1985).

[7] Baker J. E. *Reducing Bias and Inefficiency in the Selection Algorithm.* In John J. Grefenstette, editor, Genetic Algorithms and Their Applications: Proceedings of the Second International Conference on Genetic Algorithms, pp. 14–22. Lawrence Erlbaum Associates, Hillsdale, New Jersey, (1987).

[8] Barán B., Kaszkurewicz E., y Bhaya A. *Parallel asynchronous team algorithms: Convergence and performance analysis.* IEEE Transactions on Parallel and Distributed Systems, **7** : 677–688, (1996).

[9] Booker Lashon B. *Intelligent Behavior as an Adaptation to the Task Environment.* PhD thesis, Logic of Computers Group, University of Michigan, Ann Arbor, Michigan, (1982).

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

[11] Castro Y., Cerna R., Zutta K. *Algoritmos Genéticos.* Universidad Nacional de Trujillo, Escuela Académico Profesional de Informática, Trujillo - Perú.

[12] Coello C. *Introducción a la computación evolutiva.* San Zacatenco, México, (2006).

[13] De Jong A. K. *An Analysis of the Behavior of a Class of Genetic Adaptive Systems.* PhD thesis, University of Michigan, (1975).

[14] Deb K. *M.*Mhristian D. vonlti-objetive optimization using evolutionary algorithms. John Wiley & Sons, Inc., New York, NY, (2001).

[15] Goldberg, D.E. *Genetic algorithms in search, optimization and machine learning.* Addison-Wesley Longman Publishing Co. Inc., Boston, MA, USA, (1989).

[16] Lücken C. *Algoritmos evolutivos para optimización multiobjetivo.* Diciembre (2003).

[17] Michalewicz Z. *Genetic Algorithms + Data Structures = Evolution Programs.* Springer Verlag, (1996).

[18] Ronald S. *Robust encodings in genetic algorithms.* In D. Dasgupta & Z. Michalewicz, editor, evolutionaty algorithms in engineering applications, pp. 30–44, Springer-Verlag, (1997).

[19] Wetzel A. *Evaluation of the effectiveness of genetic algorithms in combinatorial optimization.* PhD thesis, University of Pittsburgh, Pittsburgh, Philadelphia, USA, (1983).

[20] Whitley D. *The GENITOR Algorithm and Selection Pressure: Why Rank-Based Allocation of Reproductive Trials is Best.* In J. David Schaffer, editor, Proceedings of the Third International Conference on Genetic Algorithms, pp. 116–121. Morgan Kaufmann Publishers, San Mateo, California, (1989).