Algoritmos genéticos en la minimización de funciones sobre intervalos pequeños

by Revista Vinculando - martes, noviembre 23, 2010

https://vinculando.org/educacion/algoritmos_geneticos_minimizacion_funciones.html

1. Introducción

La inteligencia artificial (IA) como rama de la Ciencia de la Computación se encarga de simular comportamientos inteligentes en los ordenadores. Los algoritmos genéticos son, a su vez, una rama de la inteligencia artificial basada en la teoría de la evolución de Darwin. Este tipo de algoritmos, se utiliza para abordar una amplia variedad de problemas en un conjunto de campos sumamente diverso, demostrando claramente su capacidad y su potencial. Uno de los campos especialmente abonados para el uso de los algoritmos genéticos es la optimización, debido a las características intrínsecas de los problemas que la ocupan (Buckles, Bill P. y Petry, Frederick E,1992).

En Matemática y Ciencia de la Computación el concepto de optimización se refiere a escoger el "mejor elemento" de un conjunto disponible de alternativas. En el caso más simple, esto significa resolver problemas en los que se busca minimizar (o maximizar) una función real escogiendo sistemáticamente valores de variables reales de un conjunto permitido.

La minimización de funciones puede resultar una tarea complicada en aplicaciones prácticas debido a que estas pueden tener mínimos locales, y es aquí donde los algoritmos genéticos entran a jugar un papel fundamental a la hora de enfrentar este tipo de problema. La ventaja fundamental de los algoritmos genéticos está en que son capaces de evadir los extremos locales de las funciones, pero tienen en su contra la cantidad de operaciones que realizan (Buckle s, Bill P. y Petry, Frederick E,1992).

En epígrafes siguientes se explica detalladamente cómo es posible minimizar una función de una sola variable sobre un intervalo real utilizando un algoritmo genético simple y por qué este puede ser utilizado para resolver problemas de optimización. Se analizarán además los resultados obtenidos con la solución computacional implementada.

2. Materiales y métodos

Los algoritmos genéticos son implementados como una simulación computacional en la cual una población de las representaciones abstractas de las soluciones candidata (individuos) de un problema de optimización, evoluciona hacia mejores soluciones.

¿Por qué es posible utilizar algoritmos genéticos para minimizar funciones?

A la hora de utilizar algoritmos genéticos para resolver problemas se recomienda tomar en cuenta las características de los problemas antes de intentar usar los algoritmos. El espacio de búsqueda de los problemas, o sea, sus posibles soluciones, debe estar delimitado dentro de un cierto rango. Es por ello que el caso que ocupa esta investigación se encarga de minimizar funciones sobre intervalos pequeños que están cerrados y acotados, además, la precisión de la solución se mostrará con una cierta cantidad de cifras después del punto decimal, con el objetivo de convertir en discreto el espacio de búsqueda. Lo más recomendable es resolver problemas que tengan su espacio de búsqueda discreto (ver epígrafe 2.1).

Para lograr la implementación del algoritmo genético se necesitó definir una representación de las soluciones potenciales al problema (representación de los individuos), una forma de generar un conjunto inicial de soluciones al problema (población inicial), una métrica llamada función de aptitud para evaluar cuantitativamente la optimalidad de cada solución para el problema, un operador de selección que permitiera dada una población de soluciones seleccionar los mejores individuos y reproducirlos, un operador de cruzamiento para generar nuevas soluciones del problema a partir de las mejores soluciones que se han seleccionado y un operador de mutación que produjera pequeñas modificaciones en la composición de los descendientes (Stuart J. Russell y Peter Norvig, 2004).

2.1 Tamaño del espacio de búsqueda

Para resolver computacionalmente el problema de minimizar funciones utilizando algoritmos genéticos se utilizan intervalos pequeños con el objetivo de no hacer demasiado grande el espacio de búsqueda del problema. A mayor tamaño del espacio de búsqueda del problema se necesita una mayor cantidad de iteraciones del algoritmo para obtener soluciones satisfactorias.

Es evidente que el tamaño del espacio de búsqueda depende de la precisión con que se desee obtener las soluciones. Se decidió entonces catalogar como pequeño a los intervalos de tamaño a lo sumo cien unidades con una precisión de las soluciones entre una y cinco cifras después del punto decimal. La precisión podrá ser seleccionada por el usuario de la aplicación.

Con este tamaño máximo del intervalo y con esta precisión máxima permisibles, existen a lo sumo diez millones de soluciones posibles al problema de la minimización de funciones que ocupa la presente investigación. Esto se calcula fácilmente dividiendo el tamaño máximo del intervalo entre la precisión máxima o sencillamente calculando la cantidad de veces que hay que sumar dicha precisión al inicio del intervalo para llegar al final del mismo.

2.2 Representación de la solución

Por lo general, los algoritmos genéticos trabajan con una codificación del conjunto de variables que define una solución candidata del problema que se quiere solucionar. Cada una de estas soluciones codificadas se corresponde con un individuo de la población (soluciones candidatas).

Una de las representaciones más utilizadas para codificar las soluciones de un problema es el esquema binario, en el cual cada individuo de la población es una cadena de ceros y unos donde el dígito de una posición o conjunto de dígitos de una posición a otra en

la cadena, representa el valor de algún aspecto de la solución.

En la solución al problema que ocupa la presente investigación se utilizó una codificación binaria de las variables.

Como el espacio de búsqueda del problema puede tener a lo sumo un tamaño de diez millones de soluciones posibles, entonces se necesitarán también a lo sumo diez millones de cadenas binarias para representar cada una de las soluciones existentes en el espacio de búsqueda. Este hecho implica que la menor cantidad de *bits* que se necesitan para obtener esa cantidad de cadenas binarias es 24, debido a que 2^{23} es igual a 8 388 608 y esta cantidad de cadenas resulta insuficiente para representar las diez millones de posibles soluciones. Por otra parte, 2^{24} es igual a 16 777 216, cifra que alcanza y sobra para tales propósitos. Para lograr la codificación de soluciones candidatas se utilizó la siguiente fórmula:

Donde *li* es la longitud de la cadena binaria utilizada para codificar la i-ésima variable, *xi(max)* y *xi(min)* son los valores máximo y mínimo respectivamente del intervalo donde se codifica la variable x y DV(*Si*) es el valor descodificado de la cadena *Si* (Noa Vargas, Jenny, 2008). En el caso del cálculo del mínimo de una función, la descodificación de una cadena binaria es su correspondiente valor en decimal. Esta función de mapeo permite lograr cualquier precisión de las variables si se utiliza una cadena lo suficientemente larga y también posibilita que las variables puedan tomar valores positivos y negativos.

2.3 La función de aptitud

Como se mencionó anteriormente, para utilizar algoritmos genéticos debe ser posible además, definir una función de aptitud, que indique cuán buena o mala es una determinada solución. La función de aptitud no es más que la función objetivo del problema de optimización (Goldberg, David E, 1989). Es por ello que en el caso que ocupa la presente investigación, la función de aptitud coincide con la función que se desea minimizar. Esta función evaluada en un punto del intervalo (posible solución) ofrecerá el valor de aptitud de dicha posible solución. Como el problema trata de minimizar funciones, entonces, mientras menor sea el valor de la función aptitud, mejor será la posible solución. La función de aptitud debe ser capaz de "penalizar" de cierta manera a las malas soluciones y de "premiar" a las buenas, de forma tal que sean estas últimas las que se propaguen con mayor rapidez.

2.4 Operador de selección

El operador de selección se encarga de hacer duplicados de los mejores individuos que se tienen en la población y de eliminar los peores individuos de la misma, manteniendo un tamaño constante de la población. Se hace necesario mencionar que la selección no produce nuevos individuos en la población, sino que realiza copias de los mejores. Con la aplicación de este operador se crea un estanque de apareamiento el cual es utilizado por los operadores de cruzamiento y mutación, que se encargarán de crear los nuevos individuos. Existen varias formas de realizar la selección, muchas de las cuales primeramente identifican los mejores individuos de la población, hacen copias múltiples de estos individuos y finalmente eliminan los peores individuos para ser reemplazados con las copias de los mejores. La selección proporcional de ruleta es una de estas formas y fue la utilizada en la implementación del algoritmo genético para minimizar funciones.

Según esta estrategia de selección, los individuos con mayor aptitud (*fitness*) tendrán mayor posibilidad de ser seleccionados. Se puede realizar una analogía con un juego de ruleta, donde la ruleta es dividida en N secciones donde N representa el tamaño de la población, cada sección representa a un individuo de la población y esta tiene tamaño proporcional a la aptitud de dicho individuo. Para seleccionar un individuo se hace girar la ruleta que está señalada por un puntero y cuando esta se detenga, la sección que quedó apuntada por el puntero será el individuo seleccionado.

Cada individuo tiene una probabilidad de ser seleccionado, esta es igual al cociente entre su aptitud y la sumatoria de las aptitudes de todos los individuos de la población. Luego se calcula la probabilidad acumulativa de cada individuo sumando las probabilidades individuales de cada individuo comenzando por el principio de la estructura de datos (en este caso se utilizó una lista) que los almacena. Así el último individuo de la población tendrá probabilidad acumulativa igual a uno (Noa Vargas, Jenny, 2008).

Para simular el comportamiento de este operador se hace corresponder a cada individuo un rango de probabilidad acumulativa. Para seleccionar un individuo por esta estrategia, se genera un número aleatorio entre cero y uno, luego el individuo seleccionado será el que corresponda al rango de probabilidad acumulativa al que pertenezca el número generado.

2.5 Operador de cruzamiento

Una vez que ha sido creado el estanque de apareamiento, el próximo paso es aplicar el operador de cruzamiento a las soluciones que existen en dicho estanque. Al igual que existen varias formas de aplicar la selección, existen también varias formas de aplicar el cruzamiento. El enfoque general de cada uno de los tipos de operadores de cruzamiento es seleccionar aleatoriamente dos individuos del estanque de apareamiento e intercambiar algunas de sus partes para crear nuevos individuos (Buckles, Bill P. y Petry, Frederick E.,1992).

En el algoritmo genético para minimizar funciones se utilizó el cruzamiento en un punto. Con esta técnica se determina aleatoriamente un punto de la cadena que representa los individuos seleccionados para cruzar (padres) y se generan 2 individuos (hijos) intercambiando las subcadenas derechas de ambos padres a partir de ese punto.

2.6 Operador de mutación

El operador de mutación al igual que el de cruzamiento, juega un papel importante en la tarea de buscar soluciones óptimas en un algoritmo genético. Este operador modifica partes de la cadena de representación de la solución y es muy útil para mantener la diversidad de soluciones en la población de individuos (Goldberg, David E, 1989).

Se utilizó en este caso el operador de mutación uniforme. Este operador de mutación utilizado sobre cadenas de representación binaria cambia el valor de un *bit*, de cero a uno o viceversa con una probabilidad de mutación determinada. Para implementar la mutación uniforme se recorre la cadena binaria y en cada posición se genera un número aleatorio entre cero y uno, si este número es menor que la probabilidad de mutación especificada entonces se muta ese *bit*, de lo contrario se pasa al *bit* siguiente de la cadena y se repite el proceso hasta haber recorrido cada *bit* de la representación binaria de la posible solución.

2.7 Algoritmo genético simple

La evolución comenzará a partir de una población inicial de individuos generada aleatoriamente. En cada generación se evaluará la aptitud de cada individuo, se seleccionará de la población actual múltiples individuos (según su aptitud) que serán modificados (cruzados y posiblemente mutados) para formar una nueva población que será utilizada en la próxima iteración del algoritmo. El algoritmo terminará cuando se alcance un número máximo de iteraciones suministrado por el usuario. Este último también prefijará el tamaño de las poblaciones de dichas generaciones, la probabilidad de mutación de los individuos y la semilla para la generación de números aleatorios.

Al terminar el algoritmo puede ser que se alcance o no una solución satisfactoria, debido a que esto depende del número de iteraciones escogido por el usuario.

A continuación se presentan los pasos lógicos del algoritmo genético simple (AGS) implementado:

***AGS** {*

1. *Generación de la población inicial.*
   Repetir mientras no se cumpla la condición de parada {
2. *Seleccionar por ruleta dos individuos de la población actual para la reproducción.*
3. *Cruzar en un punto estos individuos seleccionados obteniendo los nuevos individuos.*
4. *Mutar uniformemente los nuevos individuos.*
5. *Por cada nuevo individuo creado anteriormente se elimina el individuo de la población actual con menor aptitud.*
6. *Se agrega a la población actual los nuevos individuos creados.*
7. *En este punto ya se ha obtenido una nueva generación, si se cumple la condición de parada, TERMINAR (la solución es el individuo de mayor aptitud de la última generación), si no, REGRESAR AL PASO 2.*

```
 }
}
```

3. La aplicación

Se implementó una primera aproximación de una aplicación de escritorio en Visual Studio 2005 utilizando el lenguaje de programación C# que permite calcular el mínimo de funciones reales de una variable sobre intervalos pequeños utilizando un algoritmo genético.

Se programó una *interface* llamada *IFunction* con un método para evaluar las funciones reales en un punto. Con vistas a crear una nueva función, se debe programar una clase que implemente dicha interface.

Esta primera aproximación de la aplicación computacional ofrece un menú principal que le permite al usuario seleccionar un objeto de tipo IFunction que haya sido previamente almacenado de forma persistente en algún medio de almacenaje utilizando las facilidades de la serialización en el lenguaje C#.

La ventana de la aplicación brinda además facilidad para introducir los datos del algoritmo, tales como la probabilidad de mutación de los nuevos individuos obtenidos por cruzamiento, el tamaño de las poblaciones de las diferentes generaciones, el número máximo de iteraciones del algoritmo (condición de parada), la semilla de generación de números aleatorios para realizar la generación de la población inicial, el intervalo donde será minimizada la función y la precisión con que se desea obtener la solución. La figura 1 muestra la ventana de la aplicación:

Fig. 1. Ventana de la aplicación para minimizar funciones.

4. Resultados y discusión

Con el objetivo de comprobar la eficacia del algoritmo genético para minimizar funciones, se escogió la siguiente función:

para ser minimizada en el intervalo cerrado y acotado [-5;5]. Primeramente se procedió a graficar la función con vistas a tener una idea aproximada de la validez del algoritmo una vez finalizada la ejecución del mismo. Para ello se utilizó *MATLAB*: un software matemático que ofrece un entorno de desarrollo integrado con un lenguaje de programación propio (lenguaje M). Con las siguientes sentencias de lenguaje M se graficó la función F(x):

```
x=-5:0.0001:5;

y = 0.65 – 0.75*(1 + x.^2).^-1 – 0.65*x.*atan(x.^-1);

plot(x,y);
```

El gráfico de F(x) aparece en la figura 2.

Fig. 2. Gráfica de la función F(x) obtenida en *MATLAB*.

Posteriormente se procedió a realizar veinte corridas independientes del algoritmo genético para minimizar la función F(x) en las que se usaron los mismos parámetros, excepto la semilla para generar números aleatorios, la cual fue diferente en cada caso. En las corridas se utilizó una probabilidad de mutación del cincuenta por ciento, las poblaciones fueron de seis individuos, la precisión fue de cinco cifras después del punto decimal y el número máximo de generaciones fue diez. En la tabla 1 se observan los resultados de las corridas realizadas.

Tabla 1. Resultados obtenidos en veinte corridas del algoritmo genético simple para minimizar la función F(x).

De la tabla 1 se obtuvo los siguientes datos:

- El mínimo promedio en las 20 corridas fue de -0.29136
- El valor de X (punto de mínimo) promedio en las 20 corridas fue de 0.12399
- La mejor solución obtenida fue X = 0.46362, F(X) = -0.30985
- La aptitud promedio de las corridas fue de 0.23545
- Como el menor valor de F(X) alcanzado por el algoritmo fue -0.30985 y el mayor fue -0.23576, entonces la desviación estándar (espacio entre mejor y peor solución) fue de 0.07409.

Una vez realizadas las 20 corridas del algoritmo se confeccionó la siguiente gráfica con la ayuda de *Microsoft Excel*. El eje de las X representa el número de cada generación y en el eje de las Y la aptitud promedio correspondiente a cada una de ellas:

5. Conclusiones

El presente trabajo investigativo ha demostrado la utilidad y la potencia de los algoritmos genéticos como técnicas de búsqueda de la inteligencia artificial para resolver problemas de optimización, particularmente el problema de minimizar funciones reales de una variable. Al llevar a la práctica estas ideas a través de una aplicación computacional, se han obtenido excelentes resultados en la minimización de varias funciones reales.

Agradecimientos

A mi familia por el apoyo y la paciencia.

A los profesores de inteligencia artificial de la Facultad de Matemática y Computación de La Universidad de La Habana por haberme iniciado en el hermoso mundo de la asignatura.

Referencias

Libros

- Buckles, Bill P. y Petry, Frederick E. “Genetic Algorithms”, IEEE Computer Society Press, 1992, 109 p.
- Goldberg, David E. “Genetic Algorithms in Search, Optimization, and Machine Learning”, Addison-Wesley Publishing Company, 1989, 412 p.
- Stuart J. Russell y Peter Norvig Inteligencia Artificial, un enfoque moderno, Pearson Educación S.A., Madrid, 2004, 474p

Conferencias

- Lic. Noa Vargas, Jenny. Introducción a la computa ción evolutiva, 2008

Notas:

Mario Liosbel Díaz Abreu - Universidad de las Ciencias Informáticas, Carretera a San Antonio de los Baños kilómetro 21/2, Boyeros, Ciudad de la Habana, Cuba. Correo: wariol@uci.cu