Informe de Proyecto Individual

Algoritmos Genéticos en Python

Autor

Jose Pablo Trejos Conejo C07862

Universidad de Costa Rica

I Semestre

2024

Resumen ejecutivo

En este proyecto se implementó la librería *deap* como *framework* para generar un algoritmo genético que minimizara la función Rastrigin, la cual es una función que se utiliza comúnmente para probar algoritmos de optimización debido al gran número de mínimos y máximos globales que posee, con parámetros A de 10 y n de 2. Para este fin, se creó una clase llamada GeneticAlgorithm dentro del módulo GenAlgorithm. Esta clase se instanció y se ejecutó con un total de 100 individuos por generación, para un total de 5000 generaciones, dando como resultado un mínimo aproximado de 0.000022 en las coordenadas (x, y) = (-0.000199, 0.000264). Dado el mínimo global de esta función es de 0 y se encuentra en (x, y) = (0, 0), se puede observar que el valor obtenido resultó muy cercano al valor real.

Introducción:

Los algoritmos genéticos se definen como técnicas de optimización metaheurísticas o estocásticas y fueron propuestos por primera vez en los años 1970 por John Holland. Estos algoritmos, no pudieron ser probados en problemas serios de optimización hasta hace un par de décadas debido a que son computacionalmente intensivos y que, por ende, requieren de un alto poder de cómputo para su ejecución. Este tipo de algoritmos pertenecen al conjunto denominado como *evolutivos* y se basan en la idea de una búsqueda guiada a partir de un conjunto inicial de posibles soluciones denominadas población inicial, la cual irá evolucionando conforme pasen las iteraciones. La lógica detrás de este método de optimización se basa en la *Teoría de la Evolución* de Charles Darwin, la cual plantea que los individuos mejor adaptados al entorno son los que poseen mayores probabilidades de sobrevivir. En el caso de los problemas de optimización, un individuo mejor adaptado que otro, es aquel que, al evaluarse en la función objetivo, presenta valores más altos o bajos según sea el problema planteado. Es bajo este tipo de codificación que aquellos individuos que poseen mejor adaptación presentarán mayores probabilidades de ser seleccionados con el objetivo de que se reproduzcan y generen descendencia, la cual mejorará con cada iteración hasta hallar a los individuos que mejor adaptación presentaron en todas las generaciones y así encontrar el valor buscado que minimiza o maximiza la función objetivo

Metodología

Para el desarrollo de estos algoritmos se implementó la librería ***deap*** de la cual se destaca lo siguiente:

Clases y funciones implementadas:

Del módulo ***creator*** se utilizó la función ***create*** pues esta función permite generar clases según los siguientes parámetros:

- ***name***: Nombre de la clase a crear.
- ***base***: Clase de la que se desea heredar.
- ***attribute***: Los atributos que se le asignarán a la clase creada

En términos generales, con el módulo ***creator*** y la función ***create***, se simplifica la creación de clases.

Por otro lado del módulo *base* se accedió a la clase ***base.Fitness*** pues este posee los siguientes atributos de interés:

- ***values***: Tupla que contiene los valores (fitness) de calidad de cada individuo (Al optimizar funciones, este atributo contiene el resultado de evaluar al infividuos en la función).
- ***dominates***: Este atributo es un booleano que indica que si una solución es peor que otra. Se utiliza cuando se tiene un problema con múltiples objetivos.
- ***valid***: indica si el valor (fitness) de un individuo es válido.

Finalmente, del mismo modulo *base* se implementó la clase ***base.Toolbox*** pues este objeto permite registrar funciones para su llamada en diferentes secciones de código. La manera en que se registran estas funciones es mediante el método ***register*** del objeto. Este método posee los siguientes parámetros.

- ***alias***: Nombre con el que se registra la función (No es necesario que se el mismo que posee la función a registrar).
- ***function***: Función a registrar.
- ***argument***: Los argumentos que se le pasarán a la función registrada.

Procedimiento:

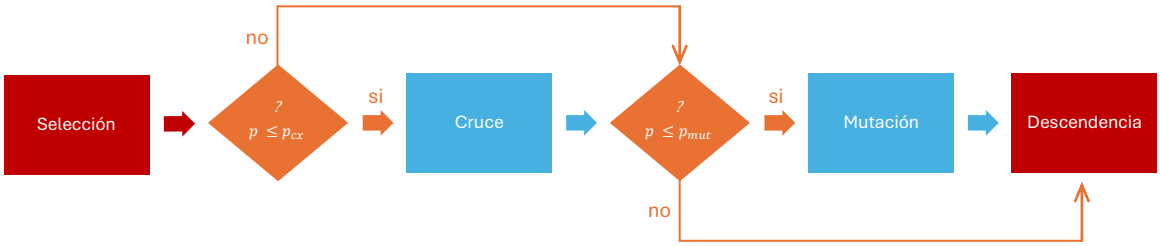
A continuación, se describe cuál es el procedimiento que realiza el algoritmo.

1. **Generación de individuos**: Inicialmente se generan individuos con genes aleatorios respetando el límite mínimo y máximo para cada valor numérico del genoma.
2. **Generación de población**: Una vez se puedan crear los individuos, se crea una población de individuos, el tamaño de esta población es escogido según las preferencias del programador.
3. **Evaluación**: Una vez que se genera la población inicial, se evalúan todos los individuos para obtener su “fitness”, es decir, su valor de evaluación. Este número es el que definirá si un individuo es mejor o peor que otro. Al evaluar a la población, se guarda al mejor individuo de todos y este no pasa por el paso siguiente, pues de esta forma, se asegura la convergencia del algoritmo.

4. **Operación genética**: En este paso, dado un conjunto de probabilidades asociadas a la selección, cruce y mutación, se selecciona un par de individuos mediante un torneo que selecciona a 3 individuos de los cuales se escoge al mejor de ellos, este proceso se repite *k* veces (los individuos se pueden seleccionar más de una vez). Una vez que se seleccionan los individuos, se procede a cruzarlos mediante el intercambio de los genes *x* e *y* entre individuos. Finalmente, el individuo resultante muta sumando una cantidad aleatoria que sigue la distribución normal.
5. **Repetición**: Al haberse ejecutado los pasos 1, 2, 3 y 4, se procede a repetir el paso 3 y 4 con la nueva población de individuos *n* veces, es decir, se crean *n* generaciones que provienen de la población inicial.
6. **Devolución del optimo aproximado**: Una vez que el algoritmo se ejecuto el número de veces que se indicó, se obtiene el individuo que mejor resultado obtuvo en todas las iteraciones y se define como el optimo aproximado de la función objetivo.

Ilustración de los operadores genéticos

Con el fin de ilustrar mejor el proceso por el cual pasa cada gen una vez es seleccionado, se muestra la siguiente cadena de flujo donde se detalla cada paso.

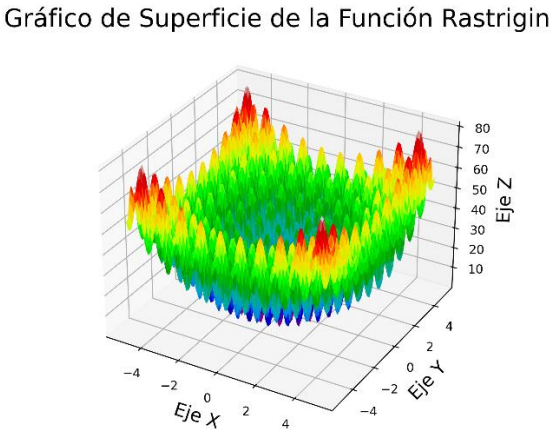


Función objetivo

Como fue mencionado al inicio, este tipo de algoritmos se centran en la optimización de una función objetivo, para este caso se escogió la función de Rastrigin la cual, como se mencionó al inicio, es una función orientada a probar este tipo de metodologías de optimización. La función en cuestión tiene la siguiente ecuación:

$$f(x) = An + \sum_{i=1}^{n} [x_i^2 - A\cos(2\pi x_i)]$$

En donde para este caso, se toma *A* como 10 y n como 2 (para poder graficarla). Cabe destacar que el mínimo de esta función se encuentra en (x, y) = (0, 0) con un valor de 0. Para los valores mencionados, la función posee el siguiente gráfico:



Resultados

Finalmente, tras correr un total de 5000 iteraciones se obtuvieron resultados muy prometedores pues en la siguiente tabla se puede observar la comparativa entre los resultados obtenidos y los valores reales de la función:

	Optimo aproximado	**Valor real**
Genoma	(-0.000199, 0.000264)	(0, 0)
Fitness	0.000022	0

Como es posible observar, los resultados obtenidos fueron bastante bueno pues para la coordenada *x* se estima un error cercano al 0.019% y para la coordenada *y* aproximado al 0.026%.

Conclusiones

Como se pudo ver en la sección de resultados, el resultado del algoritmo genético fue un aproximado bastante bueno. Considerando que este tipo de metodologías de aproximación se implementan en problemas donde hallar un valor óptimo es sumamente complicado o inclusive donde se desconoce si realmente existe uno, este tipo de resultado sería sumamente útil para una solución rápida y cercana al problema al que se podrían enfrentar múltiples profesionales.

Recomendaciones

Si bien un aproximado al valor real es deseable en problemas sumamente complejos, es posible que una solución exacta se pueda alcanzar mediate otra metodología (ejm: método gradiente), por lo tanto, es recomendable utilizar algoritmos genéticos en problemas donde una solución exacta es muy difícil de obtener o en situaciones donde el tiempo es fundamental, pues si es posible una solución exacta, pero el tiempo para su obtención es elevado, se tendría que sopesar exactitud vs velocidad, un escenario que sucede a menudo en la modelación de datos.

Referencias

Gutiérrez, D., Tapia, A., & Rodríguez, Á. (2020). Algoritmos Genéticos con PYTHON. ALPHA EDITORIAL.