Universidad Nacional Autónoma de México

Facultad de Ciencias

Tarea 1 - Reporte

Computo Evólutivo

Autores:
Diego García V.
Julio Vázquez A.

23 de febrero de 2023

Optimización continúa

Descripción y características de las funciones

Sphere

Nuestra función contiene un único óptimo global, ya que es una función *unimodal*. Se ha leído[1] *que nos ayuda a poder probar el rendimiento de un algoritmo o una propuesta de un algoritmo modificado.*

También sabemos que la función es convexa, ya que si trazamos un segmento a través de ella, ningún punto por debajo del segmento podrá sobre pasarlo.

Consideramos que al tener un **único óptimo global**, es muy difícil de optimizar, ya que aunque aumentemos de dimensión nuestra función, siempre tendrá un óptimo único.

La imagen de la función asemeja a una sábana tomada en cada punta (extremo), por una persona.

Ackley

Esta función fue presentada en las sesiones con el profesor y la ayudante. Podemos notar que tiene muchísimo mínimos locales.

También encontramos que la función es no convexa[2], por lo que nos difícilta poder encontrar los mínimos locales dada una superficie de trabajo.

Lo más curioso de está función es que su **único óptimo global** se encuentra en el punto $f(0, 0) = 0$.

Creemos que aumentar o disminuir la dimensión de la función mantiene su alta dificultad, ya que siempre tendremos múltiples mínimos locales, incluso si trabajaramos en un *espacio bidimensional* tendríamos complicaciones con encontrar el mínimo local.

Griewank

Esta función también fue presentada en la ayudantía con Malinali, pudimos observar que también tiene varios mínimos locales, pero estos son generalizados y se distribuyen de manera regular.

La dimensión nos ayuda a alterar su dificultad, ya que entre menor sea dicha dimensión, mucho más fácil es de analizar la función y de esta manera podemos encontrar la distribución de sus mínimos.

Su **óptimo global** depende de su distribución, pero al ser regulares siempre se mantiene con múltiples óptimos a lo largo de la función.

Tenth Power

Esta función es una modificación presentada por la *función Sphere*, su diferencia es que nuestra variable está elevada a la potencia 10.

Esto quiere decir que mantendrá las mismas propiedades que la *función Sphere*.

Rastrigin

La función es presentada con muchísimos minimos locales, por lo que es altamente modal, pero algo que la caracteriza es que sus localidades mínimas son distribuidas de manera regular.

La función será dificil de optimizar si buscamos justamente estos mínimos, pero altamente fácil si queremos encontrar los máximos.

La dimensión nos ayuda a alterar su dificultad si buscamos máximos, pero se nos dificultará si buscamos los mínimos por sus propiedades.

Al igual que la función **Ackley** su óptimo local se encuentra en el cuadrante 0.

Rosenbrock

La función de Rosenbrock es unimodal, por lo que el mínimo global residirá muy cerca del valle parábolico.

Es muy fácil encontrar este el mínimo local en esta función y hemos encontrado que es muy popular para las pruebas de *optimización por gradientes*.

Empeorará cada vez que vayamos creciendo nuestra dimensión.

Esquema de codificación

El intervalo de evaluación se mapea entre $0 \dots 2^n$.

Como sabemos el mapeo es una relación 1 a 1.

En una clase que contiene el intervalo, la longitud de bits, el valor en real, la cadena binaria y un arreglo de booleanos.

El *arreglo de booleanos* representa el número binario pero con valores booleanos, esto nos sirve para tener la estructura lógica del número.

Generamos el *string* como lo visualizamos y el arreglo de booleanos es como lo podemos operar (fenotipo y genotipo respectivamente).

Tamaño de búsqueda

El espacio de búsqueda es el rectangulo en $\Re^n$ generado en el intervalo, donde el intervalo es el rango de la evaluación de las variables.

Búsqueda aleatoria

Podemos acceder a la tabla de ejecuciones a través del siguiente link:
Ejecuciones

Tabla

Función	Dimensión	Mejor valor $f(x)$	Valor promedio $f(x)$	Peor valor $f(x)$
Sphere	3	$1.8388104482495684E-4$	26.228284880516664	77.41539833725864
Sphere	5	0.30499115744426286	43.69469790580357	120.65219692142588
Sphere	9	3.585023088864636	78.67145199347094	190.67068555887488
Ackley	2	1.4502824749594718	19.954401397387986	22.29555482960025
Ackley	5	7.490924234756679	20.74987604660682	22.177725738277843
Ackley	9	14.13124833015569	20.93876277689465	22.07890788446725
Griewank	2	0.07015151574184042	60.230578811087625	180.77813785576174
Griewank	5	1.4789647997231858	149.5026117850489	418.9556047562811
Griewank	9	12.65684335638531	271.3594580765735	647.434421216184
TenthPowerFunction	2	$1.042631106127349E-18$	2172550.761178825	$2.4224378689473733E7$
TenthPowerFunction	5	$9.017804034593451E-4$	5625542.1255500885	$4.973263581511871E7$
TenthPowerFunction	9	2.1035104302527907	$1.0130756505013872E7$	$6.323900466832036E7$
Rastrigin	2	0.01608369528754494	36.53407032106714	80.69454446037095
Rastrigin	5	9.94070263509439	92.48389719147004	190.34125382653684
Rastrigin	9	34.59474190113805	166.81196523677778	298.53458623656354
Rosenbrock	2	$7.112383431115087E-4$	514.2627375553499	3862.4824683402153
Rosenbrock	5	4.692007490259288	1997.6400117782835	12187.115945916024
Rosenbrock	9	54.88025818422838	3948.0571341025716	17691.564284100314

Tabla 1: Se realizo con una representación de 90 bits y $1,000,000$ de iteraciones

La tabla anterior no nos bastó para poder ejecutar nuestro programa, ya que $1,000,000$ de iteraciones no tomaban más de 10 segundos en terminar.

Decidimos probar hasta cuántas iteraciones podíamos obtener en un tiempo de espera razonable.

Por lo cuál decidimos probar con lo siguientes valores:

Dimesión: *10*

Representación de *120 Bits*

10,000,000 de iteraciones

Función	Dimensión	Mejor valor $f(x)$	Valor promedio $f(x)$	Peor valor $f(x)$
Sphere	12	2.7863999786720273	87.401815517356	213.96623422450568
Ackley	12	9.878975932264265	20.952802360972367	22.10919059881944
Griewank	12	18.84266114545478	300.97433172227164	752.1725723732053
TenthPower	12	0.7845616316002929	$1.127077071895421E7$	$6.926913511092395E7$
Rastring	12	48.31909847794876	185.26284437499814	340.9831947085734
Rosenbrock	12	52.04143073737589	4447.947615150212	21053.91293520773

Tabla 2: Representación 120 bits y $10,000,000$ de iteraciones

Optimización combinatoria

Codificación binaria

Cómo primer ejemplo de problema de combinación optimataría, lo que haremos será proponer el problema de *Gráfica bipartita.*

El cuál describimos de la siguiente manera:

Llamamos ***gráfica bipartita*** *o sólo* ***bigráfica*** *es un conjunto de vertices de una gráfica descompuestos en dos conjuntos disjuntos, del modo que no hay dos vértices de una gráfica adyacente dentro del mismo conjunto.*

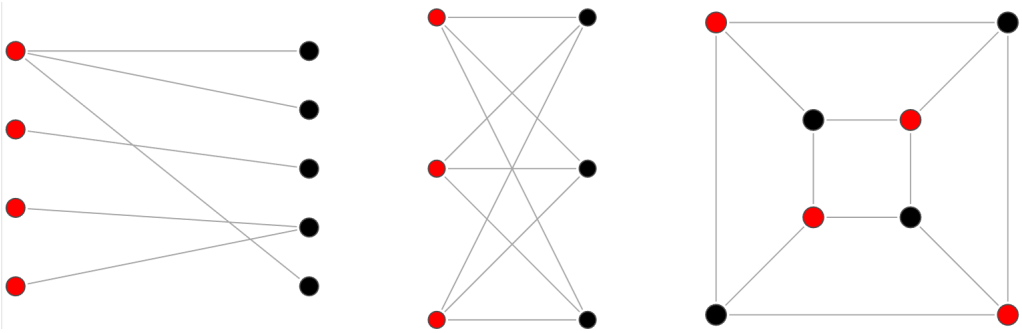


Figura 1: Ejemplo de la *k-partición* de gráficas con *k=2*

Nuestra codificación para poder representar una gráfica será sencilla, por lo cuál usaremos una matriz de adyacencias, esto en particular es excelente para nuestro problema, debido a que justamente buscamos conjuntos disjuntos dentro de G.

Por lo que la codificación será de la siguiente manera:

Sea G una gráfica. Diremos que por cada par de vértices, estarán conectados tal que cada par de vértices $(a, b) \in G$ si su evaluación en la matriz de adyacencias es 1.

Es decir:

$(a, b) \in G$ estarán conectados syss $(a, b) = 1$.

También agregamos la restricción de que un vértice no puede estar conectado consigo mismo, en otras palabras:

$(a, a) = 0$

Espacio de la búsqueda

Nuestro espacio de búsqueda lo vamos a definir como los vértices pares e impares de nuestra grafica, donde cada uno representará las distintas particiones de G.

Función objetivo

Definiremos nuestra función objetivo como la suma de las aristas con elementos de cada clase.

Es decir, tendremos dos tipos de clase.

La clase del 0 y la clase del 1.

Ya que ellas definirán el vector de n localidades con n siendo el número de nodos.

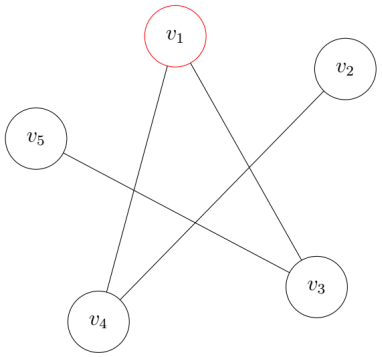
Donde si $f(E) = 0$, tendremos una gráfica que se puede bi-particionar.

Tamaño del espacio de búsqueda

El tamaño del espacio de búsqueda será toda la gráfica G, pero siempre teniendo un vértice como inicial y el cuál nunca cambiará.

Ejemplar concreto del problema

Sea G, la siguiente gráfica con el vértice v_1 marcado como fijo:



Ejemplo de solución

Elegiremos arbitraríamente dos conjuntos de nuestra gráfica y obtendremos dos vectores que nos indicarán las posiciones en donde se encuentran dichos elemento.

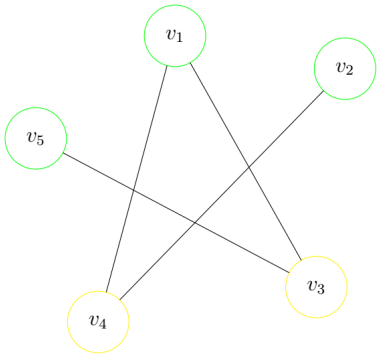
Si tomo como mi primer conjunto G_1 marcado con verde y mi segundo conjunto a G_2 marcado con amarillo obtendría la siguiente gráfica:

En la cual tengo los vértices etiquetados de la siguiente manera:

$[00000, 00001, 00100], [00010, 00011]$

Donde el primer véctor representa a los vértices: $[0, 1, 4]$ y el segundo véctor representa a los vértices $[2, 3]$

Teniendo esto, lo siguiente es hacer la función objetivo, en donde nos vamos a fijar a que clase pertenecen, si a la del 0 o a la del 1.



Verificaremos cada uno de los vértices de cada subgráfica y si tienen vértices inconexos entre ellos pertenecerán a la clase de equivalencia del 0, si tienen vértices conexos pertenecerán a la clase de equivalencia del 1.

Por lo que nuestrá gráfica de ejemplo tanto los vértices:

$v_1, v_2, v_5 \in G_1$ y

$v_3, v_4 \in G_2$

no tienen vértices conexos entre su respectiva partición de gráfica.

$\therefore$ La gráfica es bipartita, ya que la función objetivo se evalua a 0.

$f(E) = 0 + 0 = 0$

Conclusiones

En conclusión, el estudio de la optimización continua y la optimización aleatoria ha demostrado que ambos métodos tienen sus puntos fuertes y débiles a la hora de encontrar la solución óptima para un problema determinado. La optimización continua es un enfoque más centrado que usa modelos matemáticos para encontrar la solución óptima dentro de un rango de valores dado, mientras que la optimización aleatoria es un enfoque más exploratorio que usa prueba y error para encontrar la mejor solución.

En términos de rendimiento, la optimización continua tiende a ser más eficiente y efectiva cuando el problema tiene una estructura clara y bien definida, mientras que la optimización aleatoria es más adecuada para problemas más complejos y dinámicos donde la solución óptima puede no ser evidente de inmediato.

En última instancia, la elección entre optimización continua y optimización aleatoria dependerá del problema específico en cuestión y de los objetivos del proceso de optimización. Una combinación de ambos métodos también puede ser efectiva en ciertos casos, ya que pueden complementarse entre sí y proporcionar una estrategia de optimización más robusta.

En general, el estudio de la optimización continua y la optimización aleatoria ha brindado información valiosa sobre las fortalezas y limitaciones de cada enfoque, lo que puede ayudar a guiar la toma de decisiones en varios campos, como la ingeniería, las finanzas y la investigación de operaciones.

Referencias

1. Korte, B., & Vygen, J. (2006). Combinatorial Optimization: Theory and Algorithms. Springer London, Limited.

2. Cook, W. J., Cunningham, W. H., Pulleyblank, W. R., & Schrijver, A. (1997). Combinatorial Optimization. John Wiley & Sons, Inc. https://doi.org/10.1002/9781118033142

3. Ackley Function. (s.f.). Simon Fraser University. https://www.sfu.ca/~ssurjano/ackley.html

4. Sphere Function. (s.f.). Simon Fraser University. https://www.sfu.ca/~ssurjano/spheref.html

5. Griewank Function. (s.f.). Simon Fraser University. https://www.sfu.ca/~ssurjano/griewank.html

6. xloptimizer.com - Sphere function (pure Excel). (s.f.). xloptimizer.com - Optimization algorithms for Microsoft Excel. https://xloptimizer.com/projects/toy-problems/sphere-function-pure-excel#:~:text=The%20sphere%20function%20is%20one,proposed%20modification%20of%20an%20algorithm.

7. Contributors to Wikimedia projects. (2017, 3 de diciembre). Ackley function - Wikipedia. Wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/Ackley_function

8. Rastrigin Function. (s.f.). Simon Fraser University. https://www.sfu.ca/~ssurjano/rastr.html

9. Rosenbrock Function. (s.f.). Simon Fraser University. https://www.sfu.ca/~ssurjano/rosen.html