Cómputo Evolutivo
Tarea 1

Adrián García Pérez

1 Funciones de Optimización

Se definieron algunas funciones de optimización, para las cuales se requirió implementar un generador de soluciones aleatorias, implementar las 6 funciones y una hoja de cálculo con las ejecuciones.

La implementación la realicé en Python. Primero realicé la implementación de las funciones, por ejemplo:

```
def ackley(xx):
    n = len(xx)
    s1 = 0
    s2 = 0
    for x in xx:
        s1 = s1 + x**2
        s2 = s2 + np.cos(2 * np.pi * x)
    f = 20 + np.e - 20 * np.exp(- 0.2 * s1 / n) - np.exp(s2 / n)
    return f
```

que corresponde a la función:

$$f(x) = 20 + e - 20exp\left(-0.2\sum_{i=1}^{n}\frac{x_i^2}{n}\right) - exp\left(\sum_{i=1}^{n}\frac{\cos(2\pi x_i)}{n}\right)$$

Así, el programa requiere de los argumentos `funcion, dimensiones, repeticiones, ruta_archivo`, es decir, la función a graficar, de qué dimensiones deben ser las entradas, cuántas veces debe repetirse el experimeto y la ruta del archivo donde se desea tener la salida en formato `.xls`.

Así, con el archivo `makefile` y el target `all`, el programa genera los archivos con los resultados de los experimentos. Adjunto la tabla de resumen y 2 ejemplos de cada función (están completos en los archivos).

Función	Dimensión	Mejor valor	Valor promedio	Pero valor	Ejemplo en R^2
Sphere	2	14.6520282501251	25.9018459977145	36.0111810782416	[-3.81137933, -0.35414102]
Sphere	5	20.7504276005885	34.8325161759483	40.9903239337897	
Sphere	10	42.9401138286052	100.15467875008	146.735684482253	
Ackley	2	21.2228277489583	21.6556947239471	21.8557735882526	[-13.53030137, -25.88745504]
Ackley	5	20.6508624339365	21.4927513439972	21.8609655283901	
Ackley	10	21.6057077365559	21.8185770168568	22.0524377733524	
Griewank	2	4.3116858523906	63.3647825060469	93.8341506303911	[-568.13293956, 227.28644331]
Griewank	5	62.6342061589065	158.804329339949	269.008667048477	
Griewank	10	202.926980694706	312.612439402851	472.722614867985	
TenthPower	2	13493.6914577952	4260179.90725565	11663892.101772	[2.58828961, -0.76075478]
TenthPower	5	183891.007364445	4314959.34948001	10431509.996109	
TenthPower	10	3799577.51581634	11002584.2274348	25636569.5840207	
Rastrigin	2	28.3379261479405	38.0765041609696	45.5898489064636	[5.02012338, 4.12953979]
Rastrigin	5	77.7804492641838	99.8643589454726	133.499076903255	
Rastrigin	10	125.397048278757	168.76113974404	207.961347288547	
Rosenbrock	2	100.812674470182	1114.22099586376	2273.51536470989	[1.87149801, -1.26483812]
Rosenbrock	5	867.31338151633	2443.78352833847	3721.20287435866	
Rosenbrock	10	2647.76434450136	4370.13451984796	7889.65623892883	

Función	Dimensión	Semilla	$f(x)$	Tiempo de ejecución
Sphere	2	3291048657	36.0111810782416	0.01978874206543
Sphere	2	4079906556	32.0778780893419	0.010251998901367
Sphere	5	2298035088	38.9681784405673	0.010251998901367
Sphere	5	3481060450	36.4282470102716	0.010490417480469
Sphere	10	3197930589	89.5737439455996	0.014543533325195
Sphere	10	2194933518	146.735684482253	0.014781951904297
Ackley	2	2695410176	21.2228277489583	0.07319450378418
Ackley	2	3390824143	21.5362180139829	0.034809112548828
Ackley	5	4128491474	21.4153171638853	0.042438507080078
Ackley	5	3198666143	21.7281225822283	0.043630599975586
Ackley	10	2611585006	21.6858626544261	0.028371810913086
Ackley	10	3937966172	122.0524377733524	0.028133392333984
Griewank	2	3202465982	86.0646232680619	0.034332275390625
Griewank	2	2846655778	52.1472781938175	0.0152587890625
Griewank	5	3877806084	269.008667048477	0.019550323486328
Griewank	5	3416407270	200.655288638094	0.019550323486328
Griewank	10	2207873537	472.722614867985	0.033140182495117
Griewank	10	3034431066	255.44624116036	0.032663345336914
TenthPower	2	2737546183	8805391.30939879	0.021696090698242
TenthPower	2	3306348077	761488.064924166	0.012397766113281
TenthPower	5	3121768569	6163293.57050195	0.012636184692383
TenthPower	5	3700555275	4367913.81060698	0.011920928955078
TenthPower	10	2210331604	3799577.51581634	0.019550323486328
TenthPower	10	3088864204	5337598.53073473	0.020980834960938
Rastrigin	2	3198069400	34.269645971831	0.059366226196289
Rastrigin	2	3696424166	28.3379261479405	0.02741813659668
Rastrigin	5	3801425587	77.7804492641838	0.034570693969727
Rastrigin	5	2514095911	118.133066008192	0.034332275390625
Rastrigin	10	3693424872	207.961347288547	0.058650970458984
Rastrigin	10	3589525442	125.397048278757	0.059366226196289
Rosenbrock	2	2946587403	123.639573280231	0.020265579223633
Rosenbrock	2	3869944315	2092.20838445819	0.011444091796875
Rosenbrock	5	3615984070	3721.20287435866	0.021219253540039
Rosenbrock	5	4013452811	2407.01592714574	0.020027160644531
Rosenbrock	10	2534353306	7889.65623892883	0.04267692565918
Rosenbrock	10	4136566223	2647.76434450136	0.041723251342774

El procedimiento es el siguiente:

- Se obtienen las entradas antes mencionadas del programa.
- Se inicializa la salida para el archivo donde guardar los resultados.
- Por cada una de las dimensiones y el número de repeticiones, se genera una nueva semilla y se obtiene un vector de acuerdo a la dimensión.
- Posteriormente, se evalúa la función correspondiente y se miden los tiempos de ejecución.
- Se escribe en la salida los resultados de la segunda tabla.
- Finalmente, se calculan los valores de la primer tabla y se escriben en la salida.

La función utilizada es dependiente de la que se especifique en la entrada. Así, consideramos que cuando los valores que nos arrojan las funciones son menores, entonces son mejores, ya que lo tratamos como un problema de minimización. Así, nuestra función de evaluación será la propia función que busquemos optimizar.

¿Cuál sería más dificil de optimizar? La función más dificil de optimizar, es aquella que tiene más mínimos locales cercanos entre sí, debido a que es fácil caer en uno y difícil salir de él, y podríamos perder la oportunidad de encontrar un máximo global. Por el contrario, la función más fácil de optimizar sería la que tiene menos mínimos locales.

A mi parecer, dichas funciones serían Ackley y la más sencilla la Sphere.

¿La dimensión podría afectar?

A mi parecer si, debido a que encontrar valores que minimicen la función se vuelve más complicado entre más valores diferentes podemos escoger. Si tuvieramos un método iterativo que busque minimizar el problema, si la dimensión es mayor, en mi opinión podría afectar la complejidad.

Mis comentarios del ejercicio es que, a pesar de parecer confuso y complicado al principio, no resultó serlo. El proceso para resolverlo fue interesante, desde la implementación de las funciones en código hasta la generación de los resultados en tablas.

Con respecto a la representación de soluciones, en mi opinión es la representación habitual para hablar de la minimización de funciones, ya que solamente se trata de un vector que corresponde a las coordenadas donde las funciones serán evaluadas.

Con respecto al espacio búsqueda, este problema resulta ser diferente al resto, debido a que nos encontramos con que el espacio es todo el espacio real de soluciones en las respectivas dimensiones. Así, si restringimos las funciones a un rango en particular, por ejemplo la función Griewank en el rango $[-600, 600]$, entonces nuestro espacio comprenderá todo ese rango.

2 TSP

Para el conocido problema del *travelling salesman*, se inició con la lectura de un archivo donde se propone una instancia, luego se calculan las distancias correspondientes a las ciudades y después se genera una solución aleatoria para evaluarla.

La lectura del archivo se realiza recibiendo como entrada el mismo y guardando los datos que vienen en él, los cuales comprenden una sección de información del problema y otra sección con la numeración de las coordenadas correspondientes a cada ciudad.

Después de leer el archivo, se calculan las distancias utilizando la conocida distancia euclidiana, la cual implica sumar los cuadrados de las diferencias en coordenadas y obtener la raíz cuadrada.

Así, posteriormente se obtiene una solución aleatoria basada en permutaciones. Es decir, simplemente basta con dar una permutación de los índices de las ciudades que deberiamos recorrer, con el respectivo cuidado de completar el ciclo del recorrido agregando al final la ciudad con la que empezamos.

Así, obtenida la lista de indices aleatorios, realizamos la suma de las distancias de las ciudades por parejas, para obtener la distancia total del recorrido.

Así, sabremos que tenemos un buen o mal recorrido observando la distancia calculada. Entre dicho valor sea menor, mejor es la solución.

A mi parecer, es bastante interesante cómo las soluciones aleatorias pueden tener resultados tan semejantes, como se muestran en las tablas de resultados. Siendo el TSP uno de los problemas fundamentales en el mundo de la complejidad, me pareció una muy buena idea que lo retomemos en el curso para poder ver algunos acercamientos al problema desde el punto de vista del cómputo evolutivo.

Con respecto a la representación de soluciones, pienso que se trata de la más sencilla que podría darse, ya que requiere una cantidad mínima de información y somos capaces de dar soluciones sin problema.

Con respecto al espacio de búsqueda, al tratarse de un problema donde las soluciones aleatorias pueden obtenerse con cualquier permutación de ciudades, entonces considero que para n ciudades, el tamaño sería de $n!$. Así, para las instancias que tratamos en la tarea, tenemos 131, 237 y 395, siendo 131!, 237! y 395! sus respectivos tamaños del espacio de búsqueda.

3 3SAT

Finalmente, para el problema del 3SAT el procedimiento fue que, dado el número de variables lógicas y de cláusulas en la fórmula, se generara una instancia aleatoria para luego dar una solución aleatoria de esta.

En primer lugar, generamos instancias aleatorias dado el número de variables y cláusulas. Esto se consigue iterando el número de cláusulas y "repartiendo" las variables en grupos de 3, debido a que se trata del problema del 3SAT. Por ejemplo, si deseamos una instancia de 3 variables con 1 cláusula, la fórmula sería: $(x_1 \vee \neg x_2 \vee x_3)$. Si se trata de una instancia de 5 variables con 2 cláusulas, una instancia sería: $(x_1 \vee \neg x_2 \vee x_3) \wedge (x_4 \vee \neg x_5 \vee x_2)$.

Es decir, el algoritmo se encarga de generar instancias que siempre tengan el número de variables requeridas y que no caigan en casos tontos (como tener la misma variable repetida en una cláusula o que se encuentre negada y no negada en la misma cláusula). Además, se determina de forma aleatoria si las variables se encuentran negadas o no. Finalmente, la instancia creada se escribe en un archivo `.txt`, en el formato en el que se indica en la tarea.

Posteriormente, para dar una solución aleatoria, basta con darle una asignación de verdad aleatoria a cada variable. Esto se hace simplemente asignando un 0 o 1 al número de variables especificado.

Así, leemos el archivo que especifica la instancia de un problema y realizamos la evaluación. Para cada cláusula, observamos las variables que se encuentran en ella y si están negadas o no. Después, si alguna de nuestras asignaciones aleatorias para las variables es verdadera, entonces la cláusula se cumple, y esto se repite para cada cláusula.

Finalmente, la forma en la que evaluaremos nuestra solución será contando el número de cláusulas que son verdaderas con la solución; entre más cláusulas se cumplan, mejor, por lo que buscamos maximizar esta función.

En mi opinión, el tratar este problema fue igual de interesante que el de TSP, debido a su importancia en la teoría y también a la forma "tan sencilla" de dar soluciones aleatorias. Podría verse como un problema sencillo, pero se ve que con instancias de un tamaño mayor, parece ser intratable.

Con respecto a la representación de las soluciones, en vectores binarios, me parece que es una forma sencilla de hacerla, y en realidad no se me ocurre una que resulte ser mas sencilla. Se podría obtar por representaciones que sean más atractivas visualmente pero tendríamos que sacrificar la facilidad de la representación actual.

Con respecto al espacio de búsqueda, me parece que, al tratarse de asignaciones de verdad con n posibles variables, el espacio tendría un tamaño de 2^n, debido a todas las posibles asignaciones que podamos dar a las n variables.

4 Conclusiones

A mi parecer fue intersante tomar estos 3 problemas como una introducción a la materia. A pesar de que fue algo tartado generar las tablas y realizar las implementaciones de algunas funciones, fue de utilidad ver un enfoque diferente al habitual para dar soluciones aproximadas a estos problemas. Me hubiera gustado mejorar algunos puntos de mi trabajo, como una mejor explicación del código.