Cómputo Evolutivo
Tarea 5

Adrián García Pérez
315131224

Implementación

La implementación del algoritmo de Evolución Diferencial se encuentra en los archivos `main.py` e `Instance.py`, en donde como se especifica en la tarea, los vectores base se crean tomando a tres vectores aleatorios, realizando una diferencia entre dos vectores de ellos, multiplicándola por el factor de escala y sumando dicho resultado al vector base aleatorio.

Factor de escala

Experimentación para la función Rosenbrock repitiendo 10 veces, donde se hace variar el factor de escala, y los otros valores se dejan fijos: dimensión igual a 10, probabilidad de cruza igual a 0.9, población igual a 100 y el criterio de término igual a 10,000 * dim evaluaciones de función, dando como resultado la siguiente tabla:

Factor de escala	Mejor valor	Valor promedio	Peor valor	Desviación estándar
0.1	8.91	24.65	61.01	15.18
0.3	3.44	5.65	7.84	1.31
0.5	$3.49\text{x}10^{-7}$	4.32	0.0003	0.0001
0.7	0.01	29.01	47.86	17.10
0.9	5.96	8.91	11.5	1.59

Podemos notar que para un factor de escala de 0.5 se obtuvo el mejor valor de todos los experimentos, y en general los resultados con ese factor fueron mucho mejores, y por el contrario cuando el factor de escala es de 0.1 se obtiene el peor valor de todos los experimentos, aunque el valor promedio se ve superado cuando el factor es de 0.7. La tabla parece mostrarnos que un factor de escala de entre 0.3 y 0.5 resulta adecuado, aunque la cantidad de experimentos puede ser baja para poder llegar a mejores conclusiones.

Probabilidad de cruza

Experimentación para la función Rastrigin repitiendo 10 veces, donde se hace variar la probabilidad de cruza, y los otros valores se dejan fijos: dimensión igual a 10, factor de escala igual a 0.7, población igual a 100 y el criterio de término igual a 10,000 * dim evaluaciones de función, dando como resultado la siguiente tabla:

Probabilidad de cruza	Mejor valor	Valor promedio	Peor valor	Desviación estándar
0.1	$7.54x10^{-10}$	$1.11x10^{-8}$	$3.90x10^{-8}$	$1.17x10^{-8}$
0.5	20.73	30.06	41.11	6.133
0.9	42.33	227.24	310.08	100.59

En esta tabla podemos notar que con un valor de cruza muy bajo, de 0.1, el algoritmo obtiene muchos mejores resultados que con valores más altos, y parece que entre más aumentamos dicha probabilidad de cruza, los resultados empeoran, probablemente de manera "exponencial".

Experimentación con el resto de funciones

Experimentación para todas las funciones repitiendo 30 veces, donde los valores se dejan fijos: dimensión igual a 30, factor de escala igual a 0.7, probabilidad de cruza igual a 0.9 población igual a 100 y el criterio de término igual a 10,000 * dim evaluaciones de función, dando como resultado la siguiente tabla:

Función	Mejor valor	Valor promedio	Peor valor	Desviación estándar	Tiempo promedio
Sphere	0.031	0.104	0.162	0.038	15.015 s
Ackley	20.000	20.141	20.543	0.167	31.750 s
Griewank	1.186	1.416	1.774	0.157	40.270 s
Tenth	0.006	0.353	2.639	0.696	15.107 s
Rastrigin	253.691	284.732	310.086	16.025	30.201 s
Rosenbrock	30.411	38.670	47.866	4.167	27.714 s

El criterio de término sigue siendo el mismo, donde realizamos tantas ejecuciones hasta que el número de evaluaciones de la función correspondiente sea igual a 100,000 * dim, donde dim en este caso sera 30, por lo que el número total de evaluaciones permitidas será de 300,000.

Finalmente en esta tabla, podemos notar que el tiempo de ejecución promedio aumenta demasiado a comparación de los otros experimentos (donde las ejecuciones no pasaban de 5 segundos), lo que resultó en que fuera más tardado hacer el proceso de experimentación, pero ya que se realizan al menos 30 iteraciones para cada función, los resultados parecen ser más coherentes.

Por ejemplo, la función Sphere obtuvo muy buenos resultados, así como la función Tenth Power, y ambos experimentos tardaron la menor cantidad de tiempo en ejecutarse, en mi opinión gracias a que la implementación de las funciones requería menos operaciones complicadas a comparación de las otras funciones.

Por otro lado, los resultados de la desviación estándar nos muestran que en general los experimentos no tuvieron demasiada dispersión con respecto al mejor valor encontrado en cada experimento. La función de Rastrigin fue la que tuvo más dispersión en sus resultados y además no se obtuvieron buenos individuos, ya que el valor promedio fue de 284.732 aproximadamente.

Preguntas

Supón que se omite la consideración de " j == Jr" en el algoritmo clásico de Evolución Diferencial. ¿Cuál es la probabilidad de que Ui sea un clon de Xi? ¿Cuál es la probabilidad si CR=0.5 y n=20?

En primer lugar, recordemos que el vector U es creado "coordenada a coordenada", es decir que recorremos cada dimensión que lo compone asignándole un valor, donde este valor puede venir del vector base V o el vector actual X.

Para que el valor de la j-ésima posición de U sea igual al valor de la j-ésima posición de X, la desigualdad $r_{cj} < CR$ debe ser falsa y además debemos omitir la consideración de "j == Jr". Por lo tanto, la probabilidad de que U_i sea un clon de X_i, está dada por la variable aleatoria que represente n experimentos fallidos para una probabilidad $p = CR$, es decir, una distribución Bernoulli con parámetro = CR. Especificamente, la segunda pregunta implicaría 20 experimentos fallidos para un experimento que se distribuye Bernoulli con $p = 0.5$ y $q = 0.5$

¿Cómo cambiarías el algoritmo de Evolución Diferencial para que no sea elitista?

Para evitar que la población se elija de manera elitista, podríamos observar el momento en el que reemplazamos a la población de la iteración actual por miembros de la nueva población. Es decir, en el algoritmo de Evolución Diferencial que conocemos, podríamos cambiar la forma de elegir a los nuevos miembros. En lugar de elegir al mejor individuo entre el i-ésimo actual y el i-ésimo creado, podemos elegir al mejor dado una probabilidad. Por ejemplo, dada una probabilidad de 0.4 elegimos siempre al nuevo individuo creado, sin importar si es peor o mejor que el actual.

Comentarios y conclusiones

En esta tarea de Evolución Diferencial me pareció interesante esta forma de encontrar soluciones a partir del cálculo de diferencias en vectores, y ver como es muy fácil cambiar el comportamiento del algoritmo con pocos cambios en los parámetros de cruza y de escala.

Además, creo que la implementación se puede favorecer mucho utilizando un lenguaje de programación que más adecuado para el cálculo de operaciones con vectores, lo cual podría disminuir los tiempos de ejecución de los experimentos.

Finalmente, me hubiera gustado poder ver el comportamiento del algoritmo con las gráficas, pero por cuestiones de tiempo no pude hacerlo.