CC5114 - T3

Programación Genética

Nombre: Ariel Suil A.
Repositorio: asuil/CC5114-Neural
Profesor: Alexandre Bergel
Auxiliar: Ignacio Slater M.
Entrega: 23/12/20

1. Librería de Node

La librería implementada permite crear nodos y hojas con las funciones pedidas. Un `Node` puede ser una multiplicación, división segura, suma o resta; y un `Leaf` puede ser un número o una variable.

Los métodos disponibles son: `toString()`, `evaluate()`, `copy()`, `replace()`, `subcopy()`. El método `toString` retorna una representación en string del árbol; el método `evaluate` permite evaluar la expresión del árbol con las variables indicadas en un diccionario que se entrega como parámetro; `copy` crea una copia del árbol; `replace` permite cambiar un nodo y sus hijos por otro árbol; y finalmente, `subcopy` crea una copia de un árbol a partir de cierto nodo en adelante.

Se plantea definir árboles balanceados y completos, y realizar replaces que mantengan un tamaño relativamente estable, más detalles en la sección de motor.

2. Motor del Programación Genética

Para esta tarea se utilizó una versión modificada del código de algoritmo genético de la tarea 2.

2.1. Cambios

Gracias a la estructuración del motor del algoritmo genético, las únicas secciones que debieron ser modificadas fueron: el cálculo de genes a modificar, el `crossover`, y la `mutation`.

2.2. Decisiones Particulares

Al igual que con el algoritmo genético, estas secciones pueden implementarse de diversas formas, a continuación se explica la implementación de cada una en el nuevo motor.

2.2.1. Cálculo de genes a mutar

Considerando el número total de nodos que tiene un árbol y el mutation rate se toma un valor más cercano o lejano a la raíz como nodo a reemplazar utlizando la siguiente fórmula:

$$index = floor((1 - mr) * N) + noise$$

Donde mr representa el mutation rate, N el número de nodos, y noise un ruido generado aleatoriamente en un rango de 1 a N/10 (arbitrario). Los índices se distribuyen de izquierda a derecha y de la raíz a las hojas.

2.2.2. Crossover

El crossover se define como tomar la subcopy a partir del index seleccionado y remplazarla en el mismo index del individuo original. De esta forma el tamaño de los árboles se mantiene.

2.2.3. Mutación

La mutación crea un nuevo árbol utilizando una altura aproximada obtenida por el index, debido a la aproximación es posible que se obtengan árboles más grandes o más pequeños, pero gracias a la aleatoriedad del proceso, la altura promedio se mantiene en un rango estable.

3. Problemas

Para la resolución de esta tarea se presentaron dos problemas, encontrar la solución a un problema estilo "Des Chiffres Et Des Lettres" y hallar una función que pase por una cantidad arbitraria de puntos dados.

3.1. Modelación del Problema

En ambos casos, los individuos están representados por árboles de la librería ya presentada; en el caso des chiffres et des lettres existe la limitación de los números que pueden existir en las hojas; mientras que para el caso de funciones se limitó arbitrariamente al rango $[-5, 5]$ además de la variable "x".

Gracias a operaciones como multiplicar por 1 o sumar 0 se espera que el árbol balanceado y completo pueda representar otros tipos de árbol y así acercarnos a la solución sin complejizar el código.

La función de fitness de problema "dCedL" se calcula como la distancia entre la evaluación del árbol y el número esperado; mientras que para la función se calcula la suma de distancias a cada punto. En ambos casos la función fitness se ingresa en negativo dada la maximización del motor.

El algoritmo termina cuando la fitness alcanza el valor 0 o se cumplen las generaciones máximas.

3.2. Resultados

Se ejecutó el primer problema con diversas configuraciones y se encontró que dado el pequeño tamaño del árbol, era mejor tener una alta tasa de mutación, a continuación se presenta un gráfico de fitness con la siguiente configuración:

```
# goal number
N = 958
# options to build solution
N_OPTIONS = [7, 3, 50, 7, 10, 6]
O_OPTIONS = [Mult, Add, Sub]
# expected lenght of the binary representation
HEIGHT = 3
# model parameters
random.seed(528)
MUTATION_RATE = 0.8
POPULATION_SIZE = 500
GENERATIONS = 10
```

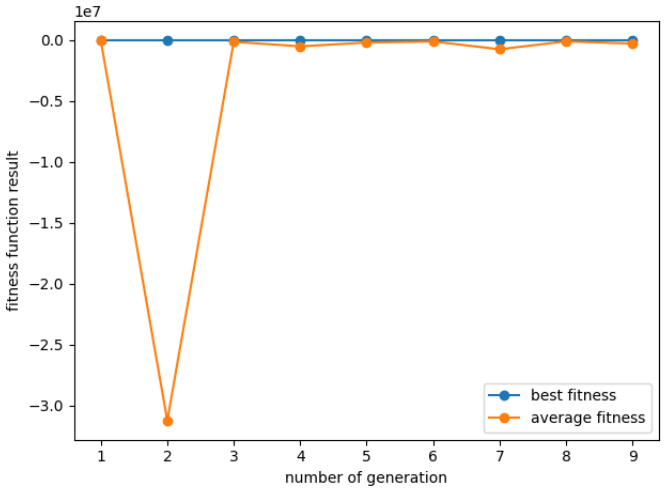

Figura 1: Fitness des Chiffres et des Lettres

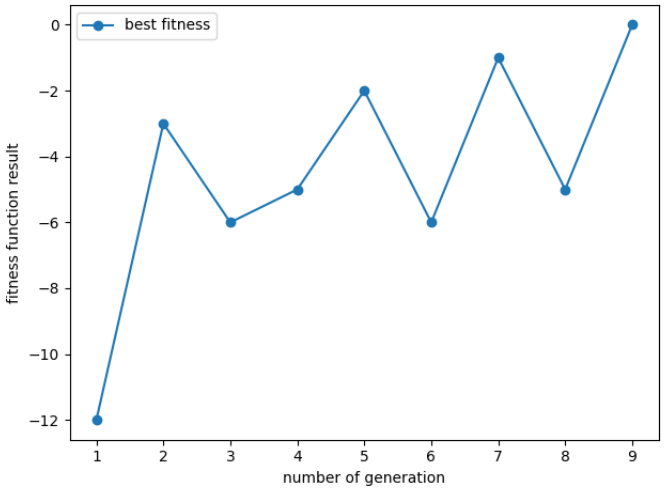

Figura 2: Best Fitness des Chiffres et des Lettres

El resultado final es la ecuación:

$$(((7 * 3) * (10 + 50)) - (((50 * 6) - (10 * 3)) - ((10 + 7) - (7 * 7))))$$

Que da como resultado exacto el valor deseado luego de 9 generaciones.

Para el segundo problema se optó por un árbol ligeramente más grande para permitir funciones más complejas, pero aún demasiado pequeño como para usar un mutation rate bajo, se adjunta un gráfico para la configuración:

```
# desired points
POINTS = [
    [3, 4],
    [6, 2],
    [7, 5]
]
# options to build solution
NODE_OPTIONS = [Div, Mult, Add, Sub]
# expected lenght of the binary representation
HEIGHT = 5
# model parameters
random.seed(528)
MUTATION_RATE = 0.8
POPULATION_SIZE = 100
GENERATIONS = 10
```

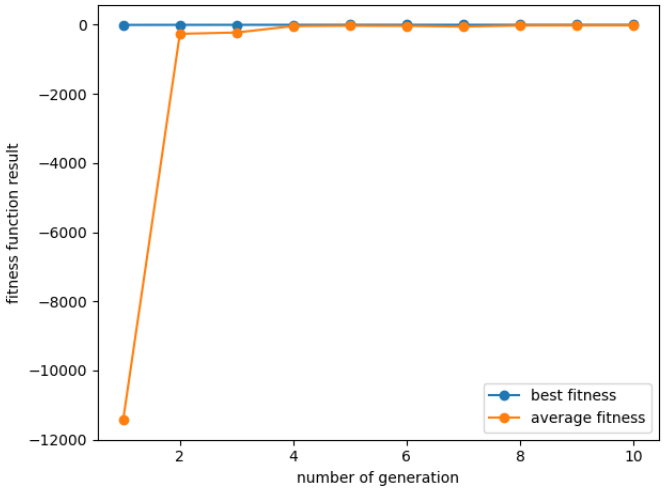

Figura 3: Fitness Función que Aproxima Puntos

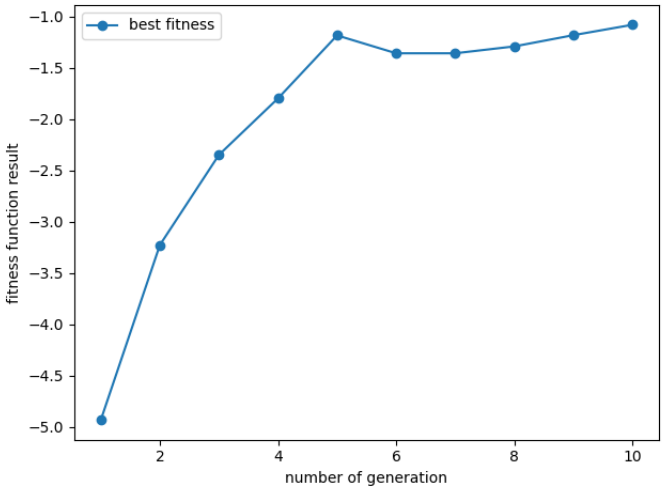

Figura 4: Best Fitness Función que Aproxima Puntos

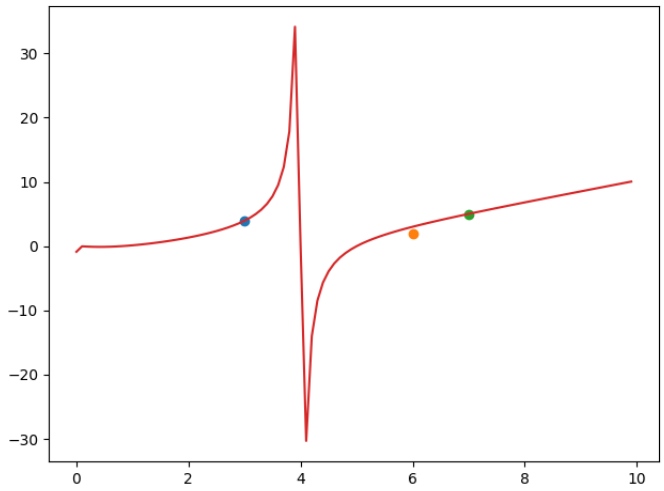

Figura 5: Función Resultado y Puntos

Función resultado:

$$(((((x + -5)\,\%(3 * -1)) * ((x * 5) - (4 * 1)))\,\% \\ (((-4 + x)\,\%(-2 - x))\,\%((5 - 3) - (-1 + -5))))\,\% \\ (((((-2\,\%x) * (3 - 5)) + ((1 + 2)\,\%(2\,\%5)))* \\ ((x + 5)\,\%(x\,\%x))) + (((3 + x) - (x - x)) * ((x + 2)\,\%(x * x)))))$$

3.3. Discusión

Para el problema de des Chiffres et des Lettres se obtuvieron buenos resultados, no se implementó que no se pudiera repetir números por lo que el resultado final posee el doble de operaciones y números que la solución sin repetir:

$$(((10 + 7) + 3) * 50) - (7 * 6)$$

Para prohibir la repetición deberíamos implementar una nueva creación de individuos la cual no utilice recursión y se maneje de manera más controlada.

Por otro lado para el problema de aproximación de puntos, se obtuvo una función bastante cercana a los puntos ingresados, pero no una que los intercepte precisamente. En el resultado final se puede observar lo planeado, existen operaciones como $() \,\%(x \,\%x)$ o $() - (x - x)$, que representan mantener un valor sin realizarle cambios, modelando un árbol que no está balanceado ni completo.