Universidad de Chile
Facultad de Ciencias Físicas y Matemáticas
Departamento de Ciencias de la Computación

Tarea 3

Programación Genética

Integrantes: Guillermo Martínez
Profesor: Alexandre Bergel
Auxiliar: Ignacio Slater

Fecha de realización: 25 de diciembre de 2020
Fecha de entrega: 25 de diciembre de 2020
Santiago, Chile

Índice de Contenidos

Índice de Figuras

1. Problema “des chiffres et des lettres”

1.1. Descripción del problema y solución

Des chiffres et des lettres es un programa de televisión Francés que prueba las habilidades numéricas y el vocabulario de los participantes. En específico, para las habilidades numéricas, se utiliza un juego en el cual dados 6 números, se debe encontrar una combinación aritmética utilizando la suma, resta, multiplicación y división para llegar a un número objetivo conocido. Además, los números elegidos no se pueden repetir.

Para resolver este problema, no se utiliza la restricción de no utilizar número repetidos. Además, se prueba con dos ejemplos, los cuales se verán a continuación.

Independiente del ejemplo, cada individuo corresponde a un árbol compuesto por nodos, en donde el padre de cada nodo es una función dentro de las permitidas por el *function set* o bien una función especial que representa el padre de un nodo terminal. Cada hijo puede ser uno de los números definidos dentro del *terminal set*, el cual corresponde a los 6 números dados para llegar al número objetivo. El hijo también puede ser una función, extendiendo así la profundidad del árbol. Con esto, una población se crea mediante la generación de Árboles de Sintaxis Abstracta (AST) que han sido construidos aleatoriamente, en base a una profundidad máxima definida.

La función *fitness* que evalúa el desempeño de un individuo entrega la diferencia entre el valor calculado por el AST y el valor objetivo, por lo que el máximo *fitness* es igual a 0.

Luego, el primer ejemplo corresponde a encontrar la combinación aritmética válida para llegar al número 459, cuya configuración es la siguiente:

- *Terminal Set*: $[25, 7, 8, 100, 4, 2]$
- *Function Set*: $[+, -, *, \%]$ (% Corresponde a la división protegida).
- Profundidad máxima de un individuo: 4.
- Tamaño de la población: 200 individuos.
- Tasa de mutación: 0.2.
- Estrategia de selección: Torneo de Tenis.
- Crossover: Se genera un único individuo por crossover.
- Estrategia de mutación: Se genera un número aleatorio entre 0 y 1, y si el resultado es menor a la tasa de mutación, se crea un nuevo individuo aleatorio y se realiza la operación de crossover entre ellos.
- Condición de término: El algoritmo termina cuando se encuentra un individuo con un *fitness* igual a 0, o cuando se alcanzan las 50 generaciones.

En cuanto al segundo ejemplo, se busca llegar al número 596 con la siguiente configuración:

- *Terminal Set*: $[10, 1, 25, 9, 3, 6]$
- *Function Set*: $[+, -, *, \%]$
- Profundidad máxima de un individuo: 4.
- Tamaño de la población: 200 individuos.
- Tasa de mutación: 0.2.
- Estrategia de selección: Torneo de Tenis.
- Crossover: Se genera un único individuo por crossover.
- Estrategia de mutación: Se genera un número aleatorio entre 0 y 1, y si el resultado es menor a la tasa de mutación, se crea un nuevo individuo aleatorio y se realiza la operación de crossover entre ellos.
- Condición de término: El algoritmo termina cuando se encuentra un individuo con un *fitness* igual a 0, o cuando se alcanzan las 50 generaciones.

1.2. Resultados

En la figura 1 presentada a continuación se puede observar un gráfico mostrando cómo evoluciona el *fitness* para el problema de llegar al número 459 durante las generaciones, ya sea tanto el mejor *fitness*, como el promedio y el peor.

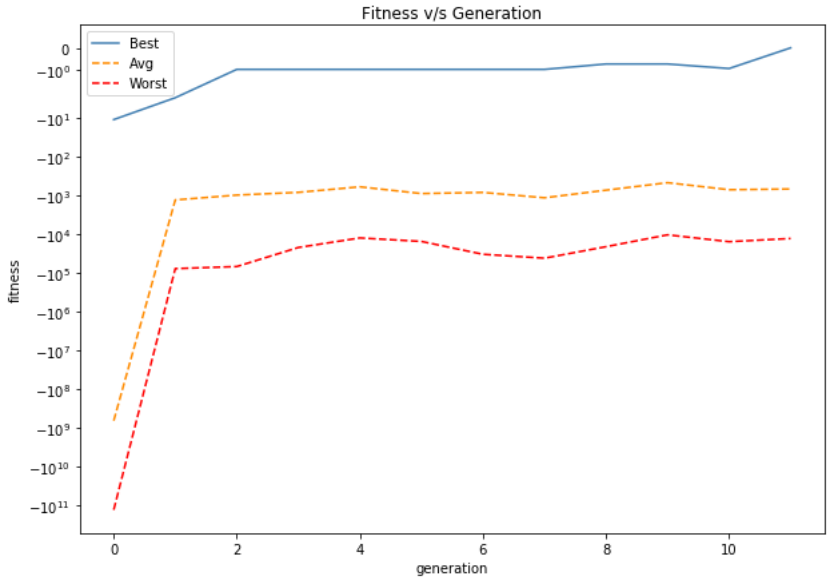

Figura 1: Evolución del *fitness* durante las generaciones para el problema de llegar al número 459.

Luego, en la figura 2, se tiene la evolución del *fitness* pero esta vez para encontrar el número 596.

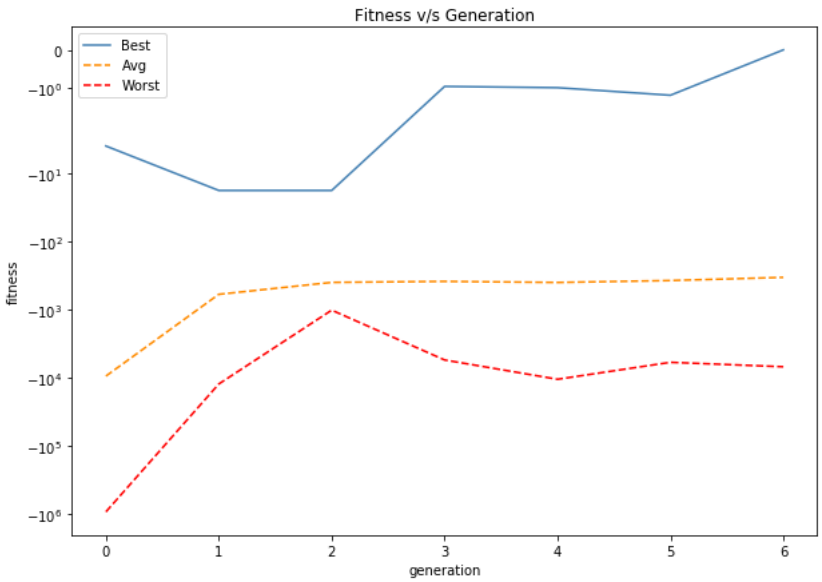

Figura 2: Evolución del *fitness* durante las generaciones para el problema de llegar al número 596.

Luego de correr el programa reiteradas veces, se llega a la conclusión de que comparar los dos gráficos anteriores no tiene sentido, ya que en definitivas cuentas el problema a resolver es el mismo. En algunas ocasiones se resuelve rápidamente (5 o menos generaciones), otras veces se demora mucho más (40 o más generaciones). Se cree que esto se debe a la libertad que se dio al poder reutilizar números, lo cual complicaría la solución de este problema innecesariamente al crear más alternativas incorrectas que correctas.

2. Encontrar función en base a puntos

2.1. Descripción del problema y la solución

El problema consiste en encontrar una función que pasa por, o cerca, de unos puntos previamente determinados.

Para este problema se realizaron 4 ejemplos distintos, pero nuevamente, independiente del ejemplo, cada individuo corresponde a un árbol compuesto por nodos, en donde el padre de cada nodo es una función dentro de las permitidas por el *function set* o bien una función especial que representa el padre de un nodo terminal. Cada hijo puede ser únicamente uno de los números definidos dentro del *terminal set*, el cual corresponde a números escogidos arbitrariamente en conjunto a una variable que llamaremos x. El hijo también puede ser una función, extendiendo así la profundidad del árbol. Con esto, una población se crea mediante la generación de Árboles de Sintaxis Abstracta (AST) que han sido construidos aleatoriamente, en base a una profundidad máxima definida.

La función *fitness* para este problema corresponde a la suma de las diferencias al cuadrado entre cada y de cada punto (que corresponde al número que se busca llegar al evaluar en x), y $f(x)$, que es algún individuo evaluado en x. Luego, el máximo *fitness* corresponde al valor 0

El primer ejemplo corresponde a encontrar la función que pasa por los puntos definidos como $[(-3,0),(-1,0),(0,0),(1,1),(3,3)]$, es decir, una función de la forma $f(x) = ReLU(x) = \max(x,0)$. La configuración para este problema corresponde a la siguiente:

- *Terminal Set*: $[0, x]$
- *Function Set*: $[+, -, *, \%, \max]$
- Profundidad máxima de un individuo: 1.
- Tamaño de la población: 100 individuos.
- Tasa de mutación: 0.2.
- Estrategia de selección: Torneo de Tenis.
- Crossover: Se genera un único individuo por crossover.
- Estrategia de mutación: Se genera un número aleatorio entre 0 y 1, y si el resultado es menor a la tasa de mutación, se crea un nuevo individuo aleatorio y se realiza la operación de crossover entre ellos.
- Condición de término: El algoritmo termina cuando se encuentra un individuo con un *fitness* mayor o igual a -0.1, o bien cuando se alcanzan las 50 generaciones.

Para el segundo ejemplo se utilizan los puntos $[(-3,-27),(-1,-1),(0,0),(1,1),(3,27)]$, que corresponde a la función $f(x) = x^3$. Se utiliza la siguiente configuración:

- *Terminal Set*: $[x]$
- *Function Set*: $[+, -, *, \%, \max]$
- Profundidad máxima de un individuo: 2.
- Tamaño de la población: 100 individuos.
- Tasa de mutación: 0.2.
- Estrategia de selección: Torneo de Tenis.
- Crossover: Se genera un único individuo por crossover.
- Estrategia de mutación: Se genera un número aleatorio entre 0 y 1, y si el resultado es menor a la tasa de mutación, se crea un nuevo individuo aleatorio y se realiza la operación de crossover entre ellos.
- Condición de término: El algoritmo termina cuando se encuentra un individuo con un *fitness* mayor o igual a -0.1, o bien cuando se alcanzan las 30 generaciones.

Como tercer función, se quiere encontrar $f(x) = x^2 + x - 3$, que pasa a través del conjunto de puntos $[(-3, 3), (-2, -1), (-1, -3), (0, -3), (1, -1), (2, 3), (3, 9)]$ y cuya configuración es:

- *Terminal Set*: $[-3, x]$
- *Function Set*: $[+, -, *]$
- Profundidad máxima de un individuo: 3.
- Tamaño de la población: 200 individuos.
- Tasa de mutación: 0.2.
- Estrategia de selección: Torneo de Tenis.
- Crossover: Se genera un único individuo por crossover.
- Estrategia de mutación: Se genera un número aleatorio entre 0 y 1, y si el resultado es menor a la tasa de mutación, se crea un nuevo individuo aleatorio y se realiza la operación de crossover entre ellos.
- Condición de término: El algoritmo termina cuando se encuentra un individuo con un *fitness* mayor o igual a -0.1, o bien cuando se alcanzan las 50 generaciones.

Finalmente, se busca la función $f(x) = x^3 - x^2 + x - 3$, un polinomio que debe pasar por los puntos $[(-3, -42), (-2, -17), (-1, -6), (0, -3), (1, -2), (2, 3), (3, 18)]$. Su configuración corresponde a la siguiente:

- *Terminal Set*: $[-3, x]$
- *Function Set*: $[+, -, *]$
- Profundidad máxima de un individuo: 6.
- Tamaño de la población: 200 individuos.
- Tasa de mutación: 0.5.
- Estrategia de selección: Torneo de Tenis.
- Crossover: Se genera dos individuos por crossover.
- Estrategia de mutación: Se genera un número aleatorio entre 0 y 1, y si el resultado es menor a la tasa de mutación, se crea un nuevo individuo aleatorio y se realiza la operación de crossover entre ellos.
- Condición de término: El algoritmo termina cuando se encuentra un individuo con un *fitness* mayor o igual a -0.1, o bien cuando se alcanzan las 50 generaciones.

2.2. Resultados

Para el ejemplo 1, el programa encuentra rápidamente la solución correcta otorgando un *fitness* igual a 0 dada la configuración ya mencionada todas las veces que se corre el algoritmo. En un inicio se creía que esto se debía únicamente al *function set* y al tamaño de la población, ya que anteriormente se tenía un *function set* limitado únicamente a la función max, con un tamaño de población igual a 500. Sin embargo, agregando funciones permitidas y disminuyendo drásticamente el tamaño de la población, el programa continúa encontrando la solución en la generación 0. Finalmente se concluye que esto se debe al *terminal set* en conjunto a la profundidad máxima permitida, ya que la cantidad de árboles aleatorios posibles es simplemente demasiado pequeño.

Con el ejemplo 2 ocurre algo muy parecido a lo mencionado anteriormente, aunque es evidente que al agregar más profundidad (necesario para resolver el problema), el programa puede llegar a demorarse un par de generaciones más. Sin embargo, en general el resultado se encuentra en la primera generación, y puede llegar a demorarse no más de 4 generaciones.

Por otro lado, el gráfico mostrado en la figura 3 es uno de los obtenidos luego de correr el programa para encontrar $f(x) = x^2 + x - 3$ en reiteradas ocasiones:

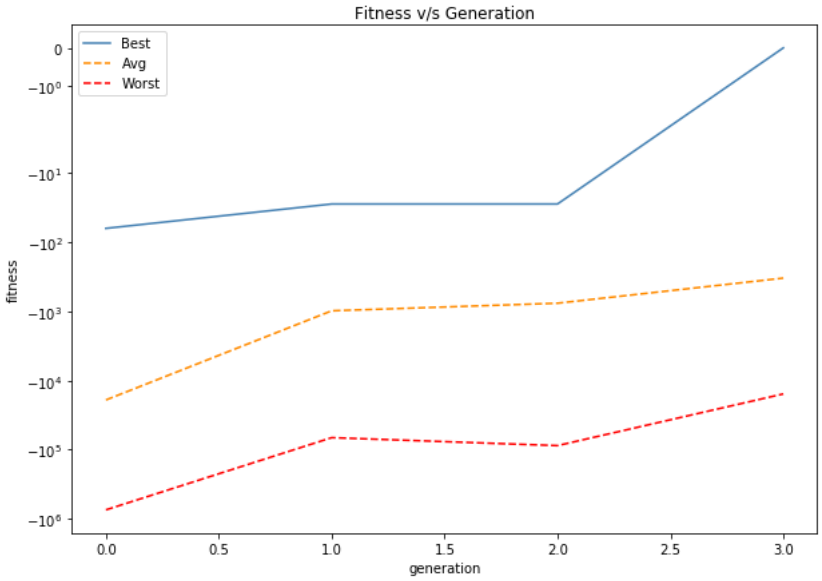

Figura 3: Evolución del *fitness* durante las generaciones para el problema de encontrar $f(x) = x^2 + x - 3$.

Si bien, tal como se puede apreciar en el gráfico, este problema es resuelto fácilmente por el programa, hay ocasiones en las que se demora una gran cantidad de generaciones (30 o más). Se cree que esto ocurre debido a máximos locales que se encuentran, ya que en general, se demora entre 0 a 3 generaciones.

Finalmente, para el cuarto ejemplo, se obtuvo el siguiente gráfico, observado en la figura 4:

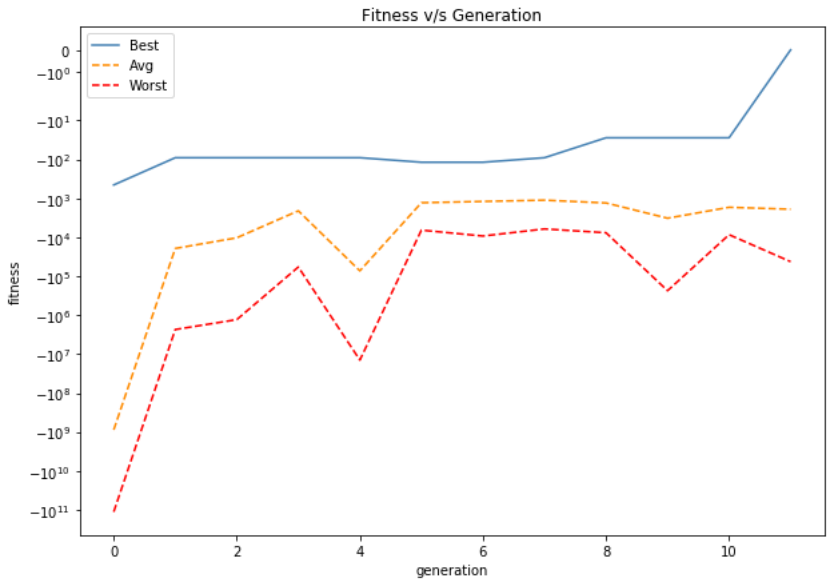

Figura 4: Evolución del *fitness* durante las generaciones para el problema de encontrar $f(x) = x^3 - x^2 + x - 3$.

Dado que para resolver este último ejemplo se requiere al menos una profundidad del árbol de sintaxis abstracto igual a 6, en un inicio era incierto el como se comportaría el algoritmo genético al intentar encontrar la función. Sin embargo, luego de probar con diversos *function* y *terminal set*, además de utilizar una tasa de mutación mayor, se logró llegar a un algoritmo genético que en general encuentra un resultado correcto. A pesar de esto, cabe mencionar que tanto el *terminal set* como el *function set* se definieron de manera que al programa le cueste lo menos posible llegar a una solución válida, al utilizar valores y funciones que a priori se sabe el programa a encontrar usa. Luego con puntos que no reflejen una función de manera explícita como los de este ejemplo el programa implementado puede fallar en encontrar una solución suficientemente buena, es decir, con un *fitness* mayor o igual a 0.1.

Link al repositorio con la resolución de la tarea: https://github.com/wiwillym/CC5114_Tarea3