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

Tarea 3

Programación genética

Alumnos:	Vanessa Gaete
	Sebastián Zapata Ascencio
Profesor:	Alexandre Bergel
Auxiliar:	Ignacio Slater M.
Ayudante:	María José Berger
Fecha de entrega:	24 de diciembre de 2020

Santiago, Chile

Propósito

El propósito de esta tarea es implementar Programación Genética para solucionar dos problemas:

- Des chiffres et des lettres
- Encontrar una función que se ajuste a puntos dados

El link al repositorio es https://github.com/VanessaGaete/Neural-Networks

Implementación de la librería de Genetic Programming

La programación genética se basa en un algoritmo muy similar al algoritmo genético pero con la diferencia de que trabaja los individuos como nodos de un árbol. Por esto, se implementa una librería con las herramientas necesarias para manejar los nodos y poder realizar las operaciones que requiere la programación genética.

En la librería se incluye una clase AbstractNode de la cual heredan otras 4 clases: Number, Add, Mult, Div, Subs y X. Cada una de estas cuatro clases se encarga de manejar lo que sucede con los distintos métodos para la programación genética dependiendo de si es un número o una operación, y en este último caso también depende de qué operación se trata. Los atributos de esta clase consisten en referencias a los hijos y al padre.

La principales funciones implementadas son copy, eval, show y replace. Dichas funciones son necesarias para poder hacer crossover y mutación.

Des chiffres et des lettres

Este es un juego que consiste en, dado un set de números y un número como solución, encontrar una función tal que utilice los números del set y que dé como resultado la solución esperada.

Para resolver este problema se generan de forma aleatoria individuos (es decir nodos), donde las operaciones son elegidas al azar y los números posibles son solo los del set dado. Los posibles nodos para el individuo son proporcionados al algoritmo mediante la variable "*node_set*", el cual en este caso contiene los nodos: Number, Add, Mult, Div y Subs. Al generar este individuo se tiene cuidado de no sobrepasar una profundidad máxima que es dada por el usuario. Los números pueden repetirse.

Luego se aplica el algoritmo de programación genética a cada generación, lo que consiste en calcular la función de fitness para cada individuo, seleccionar a los padres de la siguiente generación mediante el método de la ruleta, y así sucesivamente.

La función de fitness para este problema es inversamente proporcional a la distancia entre el valor de la función que representa el individuo y la solución. Así, mientras más alejado esté el individuo de la solución, menor es su fitness.

Resultados

Se estudió el comportamiento del algoritmo con distintos hiperparámetros, en particular, se varían la cantidad de individuos por población y el mutation rate. Se usó 10 como target y (0,1,2,3,4,5,6,7) como el set de números a utilizar.

Se prueba con mutation rate de 0.01, 0.25 y 0.5 con una población de 20 individuos. Se obtienen los siguientes resultados para el score y el porcentaje de aciertos.

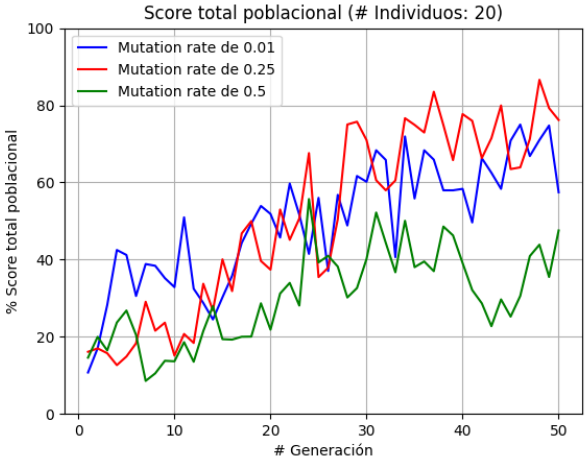

Figura 1: Solo con mutación de 0.25 se logra superar el 80 % de score poblacional. Sin embargo se puede notar que los otros iban en alza.

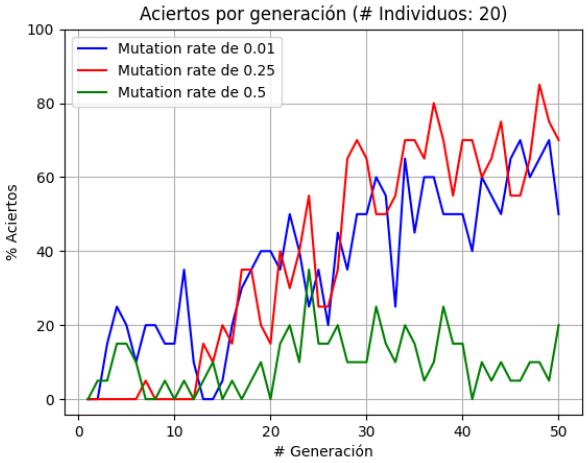

Figura 2: Con los tres valores del mutation rate el algoritmo logra encontrar la solución en algún momento. Sin embargo, hay diferencias en cuán estables son los resultados, donde un mutation rate de 0.5 hace que algunas generaciones no se encuentren soluciones a pesar de haberla encontrado anteriormente.

Por otro lado, con un mutation rate de 0.25 se comparan el *score* y el *accuracy* cuando se varía la población entre 20, 50 y 70 individuos.

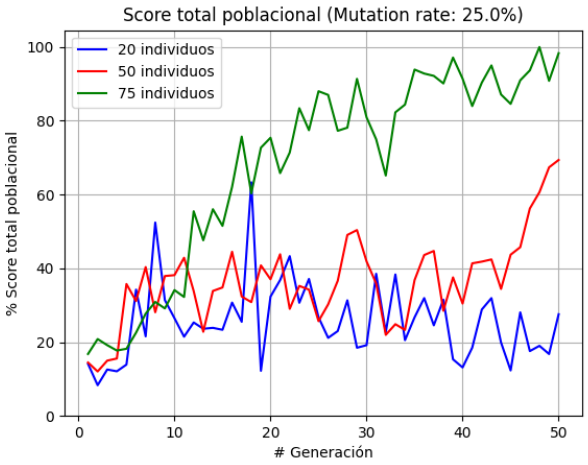

Figura 3: Tanto con 20 como con 50 individuos el score se mantiene en un rango entre 10 % y 50 %. Por el contrario con 75 individuos se alcanza hasta un 100 % de score, lo cual indica una altísima probabilidad de que el algoritmo encuentre una solución.

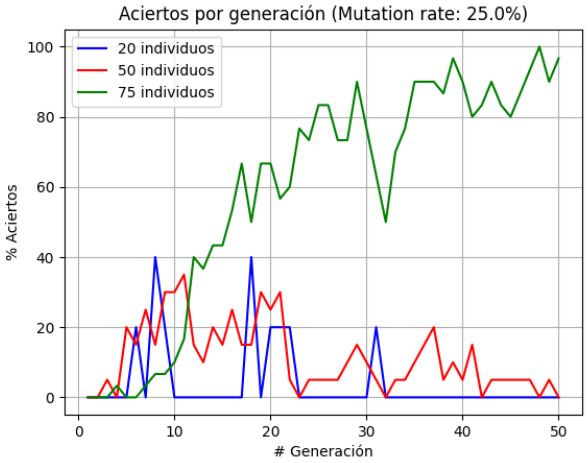

Figura 4: Con las tres opciones se logra encontrar la solución. La población con 75 individuos logra hacer que la mayoría de sus individuos sean soluciones correctas, mientras las otras solo encuentran algunos y en algunas generaciones ni siquiera encuentran.

Encontrar una función

La idea de este problema es encontrar una función de una variable tal que pase por los puntos dados.

El set de nodos proporcionado para este problema es igual al del problema anterior pero se le añade el tipo de nodo X. Este representa una variable en una función.

Al igual que para Des chiffres et des lettres, se genera una población inicial de individuos de forma aleatoria, sin embargo en este caso no se otorga un set de números con los cuales armar la función, por lo que se eligen números aleatorios entre -5 y 5. A diferencia del problema anterior, en este caso se pueden crear nodos X.

Luego de esto se aplica el algoritmo para la población por la cantidad de generaciones especificadas.

En este caso el fitness también es inversamente proporcional a la correctitud del individuo. Para calcularlo, por cada punto dado en el *points_list* se evalúa al individuo reemplazando sus nodos X por el valor en "x" del punto, luego se calcula la diferencia entre el valor del individuo y el valor de "y" en el punto. Los resultados para cada punto se suman, correspondiendo al score final.

Resultados

Para estudiar la influencia de los hiperparámetros en los scores y aciertos de la población se comparan algoritmos con 50 generaciones y que buscan una ecuación que pase por los puntos: (1,1), (2,4), (3,9) y (4,16).

Se compara el comportamiento del algoritmo para mutation rate de 0.01, 0.25 y 0.5 y con 20 individuos por población.

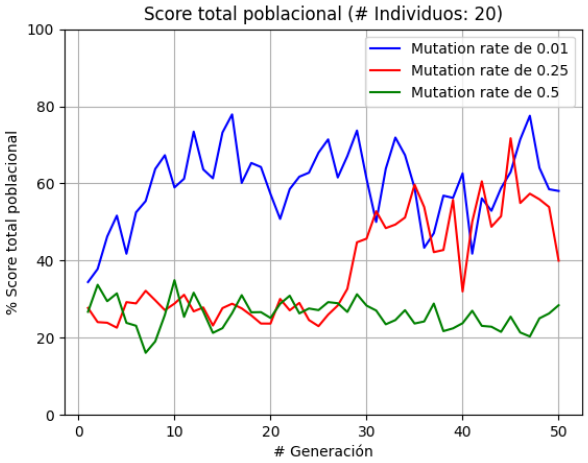

Figura 5: con un mutation rate de 0.5 no se logra salir del rango 20 % - 40 % de score. Los otros dos alcanzan un score similar con el paso de las generaciones. La probabilidad de encontrar la solución con un mutation rate de 0.5 es la más baja.

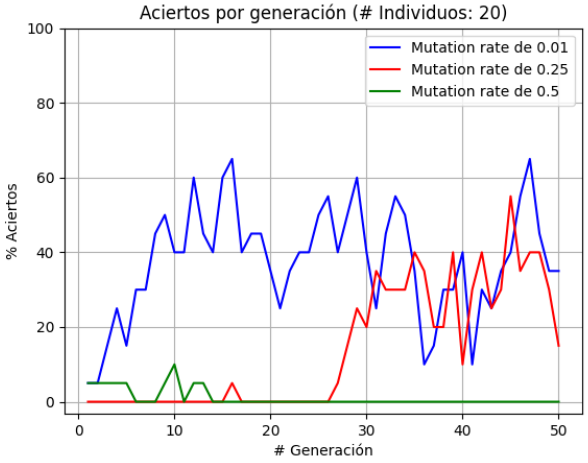

Figura 6: La probabilidad de encontrar la solución con un mutation rate de 0.5 es baja. Con los otros dos valores se encuentra la solución en varias generaciones, en particular con 0.01 se encuentra en casi todas.

También se estudia la influencia del tamaño de la población, variando entre 20, 50 y 75 individuos. Se fija un mutation rate de 0.25.

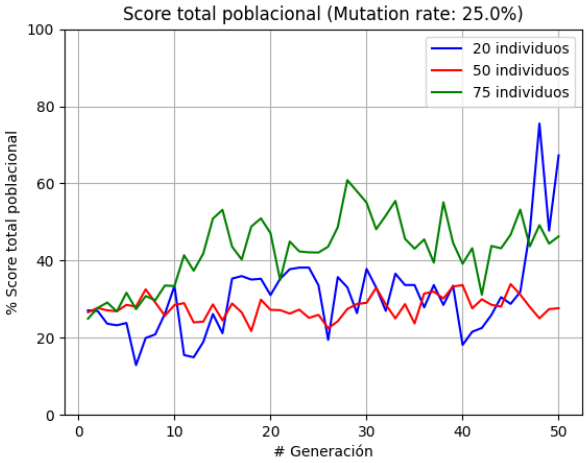

Figura 7: Se logran scores bastante parecidos. Solo el caso con 20 individuos supera el 60 % casi en las últimas generaciones, por lo que no se sabe si se mantiene estable luego de eso. Así, el algoritmo con 75 individuos es el que tiene la mayor probabilidad de encontrar la solución.

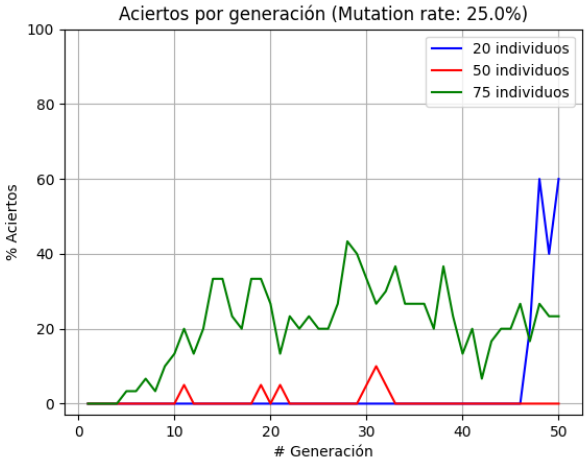

Figura 8: Con 20 individuos por generación se encuentra una solución casi en 50 generaciones a pesar de que el score fuese más alto que con 50 individuos en algunas generaciones, lo que indica que habían varios individuos cercanos a la solución pero aún así ninguno correcto. Con 75 individuos se encuentra casi en 5. Con 50 individuos rara vez se encuentra la solución.

Conclusiones

En los ejercicios anteriores, era muy común esperar que muchos individuos convergan hacia una misma solución para un problema dado, aunque hubiese más de una posible. Sin embargo, gracias a estos ejercicios es posible encontrar nuevos aspectos que estaban ocultos a simple vista, uno de estos es la sensibilidad del fitness entre un cromosoma y otro. En problemas como encontrar una palabra o dar con la representación binaria de un número, la alteración de un solo gen no variaba demasiado el score de un individuo, pues era una variación pequeña, pero al tener nodos que generan expresiones numéricas, pasar de una suma a una multiplicación puede suponer un gran cambio en el score si este se basa en la distancia entre el resultado esperado y el obtenido.

La primera conclusión se hace notar al observar los resultados de ambos ejercicios, pues, a pesar de encontrar una generación con más de la mitad de individuos aceptados como solución, estos pueden bajar considerablemente en cantidad en las próximas generaciones, y esto tiene sentido con el hecho de que, esta vez la cruza sin mutación entre dos individuos considerados solución, no necesariamente darán resultado a otro individuo que solucione el problema, pues la combinación de su material genético altera muchísimo al individuo final y por lo mismo, no es eficiente tener demasiados individuos ni mucho menos tener grandes valores para el ratio de mutación como se aprecia en los resultados.

Por lo expuesto, se concluye que para estos ejercicios, no es necesario avanzar en las generaciones y cruce entre individuos una vez hallada una o varias soluciones, pues no tiene sentido si ahora no se espera converger hacia una misma solución, se propone en futuras iteraciones que este programa genético finalice antes de las generaciones pedidas si es que encuentra solución.

Por último, existe una curiosidad para el ejercicio 1 cuando se busca el número 0. Dado que al multiplicar cualquier número por cero, da cero, es muy fácil para una población encontrar expresiones que resulten en este valor, esté o no el mismo agregado además al set de números posibles. Esto además permite concluir que las opciones que el programa puede tomar para la generación de sus individuos, así como la elección de la solución a buscar, influyen fuertemente en la rapidez para encontrar esta última. A nivel de experimentación es posible jugar con estos valores y limitar al programa en cuanto a sus elecciones, pero en programas que busquen dar soluciones a problemas más reales y que sean difíciles de encontrar manualmente, es mejor poner más atención en estos parámetros, pues pueden influir demasiado en la ejecución final en busca de una solución.