**Luis Enrique Zamudio Cervantes**
**luis.zamudio90@gmail.com**
**307293136**
# Práctica 4

## 1. Introducción: Programación Genética

La programación genética es una técnica de inteligencia artificial que nos sirve para explorar espacios de búsqueda muy grandes, es muy similar a un algorítmo genético, pues trabaja con los mismos elementos, una población y selecciona a individuos de esa población para cruzarlos, mutarlos y evaluarlos, tal como un algorítmo genético.

Como es de imaginarse, al tener mutación y cruza, también se debe de contar con una probabilidad de mutación y una probabilidad de cruza, también maneja el concepto de elitismo, y natualmente se cuenta con un método de selección para la recombinación de individuos.

¿Entonces cuál es la diferencia con un algorítmo genético?
La diferencia está en que un algorítmo genético nos modela soluciones, es decir nuestos individuos de la población son codificaciones de soluciones posibles al problema, dependiendo de esa codificación es que vamos a evaluar que tan bueno e el individuo, sin embargo en la programación genética no nos va a interesar tanto la solución del problema dado, pues muchas veces ya conocemos la solución, pero quisieramos tener una manera de calcularla, por lo que encontar la manera es en lo que se enfoca la programación genética.
Entonces podemos pensar que nuestros individuos son codificaciones de secuencias de instrucciones, vamos a llamarles programas por comodidad del lenguaje.

La idea entonces de programación genética es tener un conjunto de individuos que modelen diferentes programas y el mejor individuo es el que logre resolver el problema, dentro de los que logren resolver el problema, van a destacar sobre los demás, los que lo logren de una mejor manera, ya sea más rápido o con una función que computacionalmente hablando no sea costosa, o que sea más corta su descripción, depende el problema y de los programas podremos discernir entre uno u otro.

Ya que dijimos que programación genética es básicamente tomar un conjunto de programas, recombinarlos y modificalos para que logren hacer una función específica, nos preguntamos cómo hacerle para modelar dicho programa y la manera en que lo hace la programación genética es muy simple, se trata de diseñar una gramática que nos permita realizar expresiones válidas para ciertos operadores que vamos a definir y para ciertos conjuntos de valores a utilizar, es decir es muy parecido a definir un pequeño lenguaje de programación, sus expresiones y sus elementos.

Como vamos a modelar un micro lenguaje de programación, nos surje la necesidad de definir una gramática, por lo tanto nos interesan 2 conjuntos principales en particular, estos conjuntos son el conjunto de reglas de producción y símbolos terminales, esto lo podemos modelar de una manera muy sencilla con 2 conjuntos, el conjunto de símbolos terminales, al cual nos vamos a referir por "terminales z el conjunto de operadores que vamos a poder aplicarle a los elementos de nuestro espacio, a este conjunto le vamos a llamar "funciones", de manera que ya podemos ahora escribir todas las exresiones válidas para nuestro pequeño lenguaje.

## 2. Contexto y objetivos de la práctica

El principal objetivo de esta práctica es lograr encontrar programas que modelen funciones específicas, es decir que dado un elemento logren responder cosas sobre ese elemnto, por ejemplo si cumple alguna propiedad, o si es posible encontrar una función que nos acote un problema o que nos modele una solución.
En este caso se trata de encontrar soluciones particulares de problemas muy particulares a los cuales podríamos tener una solución desde el inicio, pero nos interesa encontrar una mejor o en su defecto lograr que con programación genética y unos cuantos individuos básicos al juntarlos, logremos lo deseado.
Problemas:

- Paridad Par:
  Se dice que una cadena de bits (que representa un estado para variables booleanas), tiene paridad par, si cuenta con un número par de unos en ella.
  El problema a resolver es que dada cualquier cadena de bits, digamos si la cadena tiene paridad par o no.

- Suma a 10:
  El problema es muy simple, se trata de encontrar un programa que sumando y/o multiplicando números encuentre una operación tal que su resultado sea 10.

- Regresión:
  El problema de regresión es un tanto común en el área de cómputo científico, la idea de regresión es encontrar una función que nos modele resultados para un conjunto de elementos, esto con el menor número posible de errores, en este caso particular es que dado un conjunto de puntos en el plano, encontrar una función que pase lo más cerca posible de todos estos puntos en el conjunto. Esto se va a modelar con una tabla que por una parte en el dominio tiene los valores que tiene X y en la otra los valores que le corresponden en la imágen, es decir una tabla de X|F(X).

Notemos que los 3 problemas se tratan de encontrar funciones que se aproximen (o encuentren) las soluciones exactas, esto quiere deccir que mientras menos errores mejor serán nuestras funciones, lo cual nos hace pensar que sería una buena idea tener una función de fitness que se base en los errores que tenga la solución, es decir que nuestro fitness va a modelar errores, por simplicidad, es más fácil contar e ir sumando errores que tomar todos los fitness y luego normalizarlos, por lo que vamos a decir que un individuo va a ser mejor que otro si tiene un fitness más bajo (pues eso quiere decir que tiene menos errores), por lo tanto quisiéramos encontrar individuos con fitness 0, es decir que encuentren una solución exacta del problema.

## 3. Restricciones y modelo de los problemas

1. Paridad Par:
   Este problema lo vamos a modelar de manera muy simple, tomaremos un conjunto de variables booleanas, a los elementos de este conjunto son a los que debemos de operar para determinar la paridad de una cadena, como una cadena modela estados de las variables, entonces vamos a hacer un emparejamiento de lugares en la cadena con las variables booleanas, es decir si tenemos una variable A, el estado de esa variable está modelado por el bit en la posición 0 de la cadena, lo mismo para las demás variables con otras posiciones, de manera que al tener un conjunto con k variables, tendremos que generar $2^k$ cadenas booleanas, para representar todos los estados posibles para las K variables.
   Como es una operacion booleana la que tenemos que encontrar, nuestro conjunto de funciones van a ser operaciones booleanas.
   En este caso vamos a encontrar funciones que nos den la paridad par para diferentes tamaños de cadenas.
   Con lo antes dicho, llegamos a que el conjunto de Funciones es: $Funciones = \{AND, OR, XOR, NOT\}$ y el conjunto de terminales será variado dependiendo de la longitud de la cadena de la que necesitemos determinar su paridad, en particular se usó el conjunto: $Terminales = \{A, B, C, D, E, F, G, H, I, J\}$ recortándolo según la longitud que deseemos de la cadena. La longitud se variará en diferentes pruebas con 2,3,4,7 y 10 variables.

2. Suma a 10
   El modelo de este problema es mucho más simple, los conjuntos que se utilizan son: $Terminales = \{0, 1, 2, 4\}$ y el conjunto de funciones lo cambiaremos, en la primera prueba el conjunto será $Funciones = \{+\}$ y en la segunda $Funciones = \{+, *\}$

3. Regresión:
   Este problema lo vamos a modelar a partir del conjunto $Terminales = \{0\,, 1\,, 2\,, 3\,, 4\,, 5\,, 6\,, 7\,, 8\,, 9\,, 10\,, X\,, R\}$ donde X representa al elemento X del dominio de la función o a la coordenada x del punto en el plano (cualquiera de los 2 enfonques con el que lo queramos tomar, Yo prefiero los puntos en el plano) y R representa un valor aleatorio(pero fijo) en el intervalo $[0\,, 1]$. En cuanto a los operadores serán: $Funciones = +\,, ?\,, ?\,, /\,, SQRT\,, SIN\,, COS\,, EXP\,, LN$

# 4. Implementación

El modelo teórico ya quedó completamente explicado, pero ¿Que hay de la parte práctica?.

En realidad pasar del modelo teórico a la práctica se hace de una manera muy intuitiva, como tenemos conjuntos de operadores y de operandos, quisieramos validar que en un programa válido aplicaramos el operador OP1 del conjunto de cunfiones a un elemento E1 del conjunto de terminales, pero más aún, queremos poder hacer operaciones de operaciones de elementos, es decir poder hacer OP1(OP2 E1) donde OP2 es un operador de aridad y recibe a un elemento E1, pero más importante aún, que los operadores deben de cumplir con la propiedad de la cerradura, es decir al aplicar el operador OP2 a un elemento E1 debe de regresar un elemento del espacio, y más aún, si a ese resultado le vamos a aplicar el operador OP1, entonces debemos de validar que el tipo del valor de retorno de OP2 sea del tipo que recibe OP1, por lo qu E1 pasa de estar en los terminales a poder ser cualquier elemento del espacio, tal que OP1 lo puede recibir como parámetro formal.

Todo esto suena muy complicado, pero en programación es fácil e intuitivo modelar todo esto con un árbol, en este caso con nodos potencialmente binarios, es decir que a lo más tendrán 2 hijos, algunos operadores, naturalmente sólo pueden tener 1 hijo, como los operadores EXP, LN, SIN y COS en el problema de regesión o el operador NOT en el de paridad par.

En cuanto al dominio no tendremos mayores problemas (al menos de tipos) pues si operamos con esos conjuntos y operadores, todos caen en el mismo espacio, sin embargo, debemos de andarnos con cuidado, porque los dominios y ocntradominios de los operadores se nos pueden salir de las manos, por ejemplo, el operador LN (modela la función $ln(x)$, donde x lo recibe como parámetro) no puede recibir números negativos, por lo que tenemos que tener cuidado con eso, si no, el resultado se indetermina, los mismo para cada uno de los casos especiales, pero eso lo podemos manejar fácilmente en programación teniendo cuidado a la hora de la evaluación de los individuos.

Sin más rodeos y ya con más detalle, todo esto lo vamos a mapear a un árbol binario que como nodo, guardarán un valor, puede ser una función o un terminal, de ser un terminal no tiene hijos y de ser una función, tendrá como hijos un número de nodos igual al número de parámetros que reciba, esto quiere decir que también debemos cuidar la aridad de nuestros operadores para la asignación de hijos.

Si nuestros individuos serán árboles, entonces necesitamos una representación en forma de cadena para los individuos, de manera que lo más simple es representar un árbol con alguno de sus recorridos, Yo elegí el recorrido inorden, así el orden induce un orden para los nodos, ese mismo orden es el que posteriormente vamos a utilizar para la mutación y la cruza.

Con esto dicho, ya podemos entrar en detalles sobre la programación genética que operadores genéticos vamos a utilizar y los conjuntos de elementos para nuestro árboles.

## 5. Operadores de la programación genética

- Mutación:
  La mutación que utilizamos para resolver es de hecho muy simple, se toma un nodo aleatorio de nuestro individuo que es un árbol y vemos la altura que tiene, entonces vamos a sistituírlo por un nodo de una altura similar al que removimos, por similar nos referimos a un nodo que tenga altura entre 1/2 y 3/2 de la altura del eliminado. A esta mutación se le conoce como mutación de tamaño justo.
  Vamos a mutar elementos de la población con probabilidad de .2, es decir un %20 de la población va a mutar, esto hace más grande el espacio de búsqueda una vez que tenemos una población en particular

- Cruza:
  La cruza es igual de simple que la mutación, se toma un nodo de manera aleatoria del árbol1, así mismo con el árbol2 y los intercambiamos, con una pequeña restricción, suponiendo que el nodo aleatorio que tomamos es la raíz del árbol1, si vamos a insertar la raíz del árbol en el lugar del nodo que quitamos del árbol2, pero no vamos a cambiar la raíz del árbol1 por el nodo del árbol2.

## 6. Descripción de las pruebas

Todas las pruebas que se hicieron fueron iguales, en el sentido de que los operadores genéticos son los mismos, al igual que las probablidades de mutación y cruza, el método de selección, las características de la población inicial, y cada problema con sus respectivos conjuntos de Funciones y Terminales descritos arriba.

La configuración inicial es:

- Poblacion inicial con 100 individuos, probabilidad de mutación de 0.2 y probabilidad de cruza de 0.8.
- La población inicial está conformada de individuos creados con el método de rampeo, los individuos tienen altura máxima de 6(Esto sólo en la generación de la población, más adelante esa altura aumenta).
- La mutación y la cruza son las descritas anteriormente.
- El método de selección para la cruza es el de Vasconcelos.
- El programa utiliza elitismo total
  Cabe destacar que el elitismo no afecta a nuestras pruebas de esta práctica, porque si almacenamos los mejores individuos de todas las generaciones, pero lo que en realidad nos interesa es ver que tan rápido alcanzamos individuos óptimos, es decir no tomamos en cuenta a los individuos que guardamos en el elitismo, sólo a los de la generación actual, para las gráficas y la ejecución del programa. Al final en el archivo de texto que se genera, el último renglón es el mejor individuo que se encontró en toda la pueba.

# 7. Pruebas y Resultados

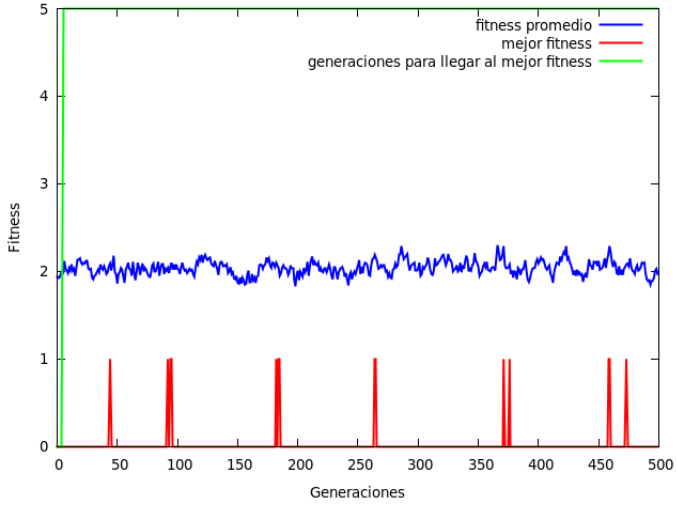
En los requerimientos de la práctica se piden al menos 5 corridas de cada prueba y las hice, anexo también los datos y las gráficas de estos, sin embargo no pondré las gráficas de las 5 corridas, pues son muchas gráficas y es poco amigable para entender los resultados de las pruebas.

En la primera gráfica de cada inciso se pueden ver los valores que representan a los fitness.
Las líneas verdes de la gráfica, representan las generaciones en las que se encontraron mejores individuos, cada que se encuentra un mejor individuo (desde la primera generación) el valor de la línea verde aumenta.
Las líneas Azules representan el promedio del fitness de la población.
Las líneas Rojas representan el mejor fitness de la población

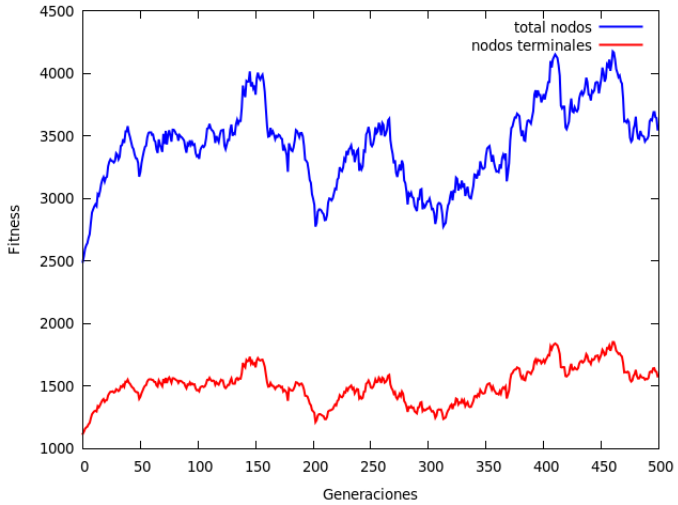
En la segunda gráfica veremos la relación de total de nodos(en azul) y nodos terminales(en rojo) en toda la generación.

1. PARIDAD PAR

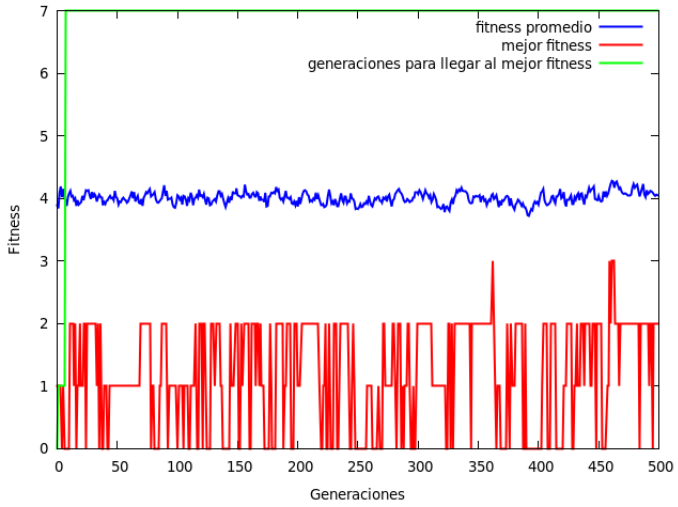
   *a*) 2 Variables



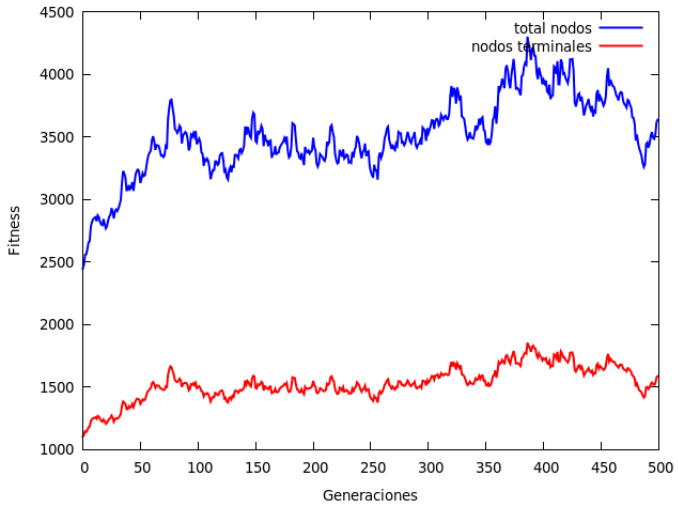

En la gráfica podemos ver que desde el inicio se encuentra el óptimo en la generación 5 aproximadamente, y también que casi siempre se encuentra la función sin errores y si hay errores a lo más es 1, también vemos que el fitness promedio es relativamente estable con media de 2 errores



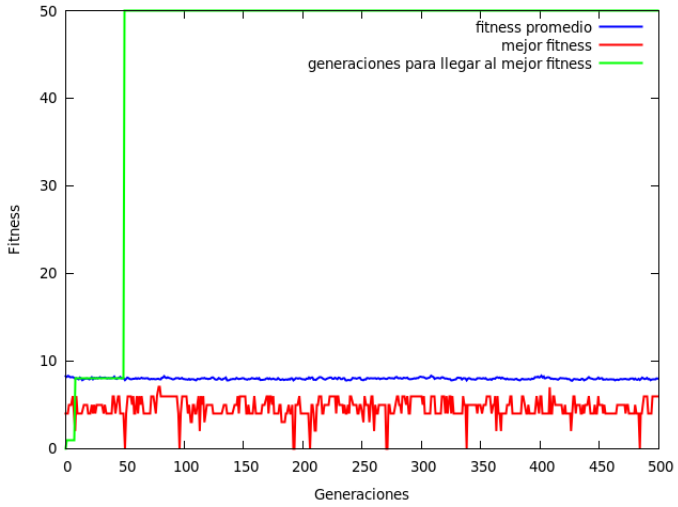
*b*) 3 Variables



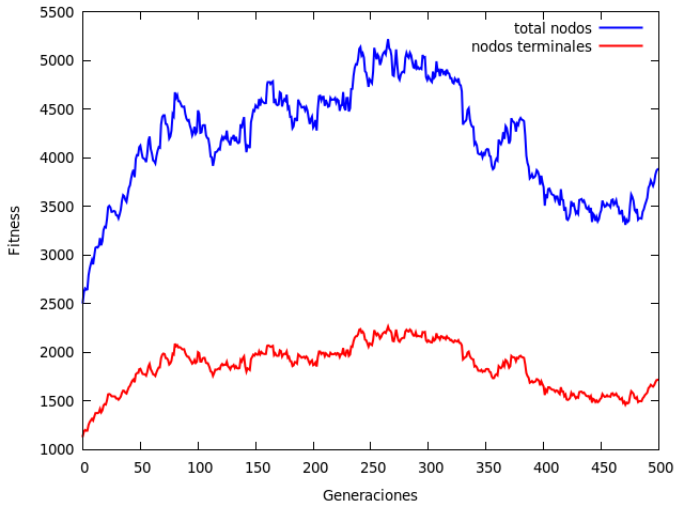

En comparación con la anterior, igual se encuentra el óptimo más o menos en el mismo tiempo, pero el mejor individuo de la población varía su mejor fitness entre 0 y 2



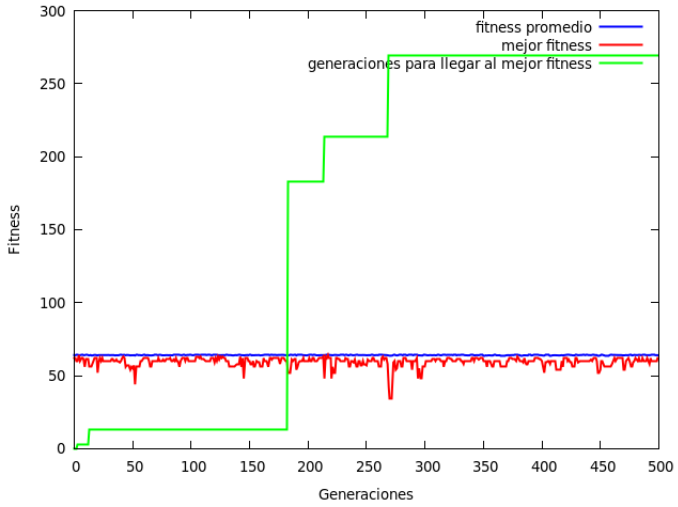
*c*) 4 Variables



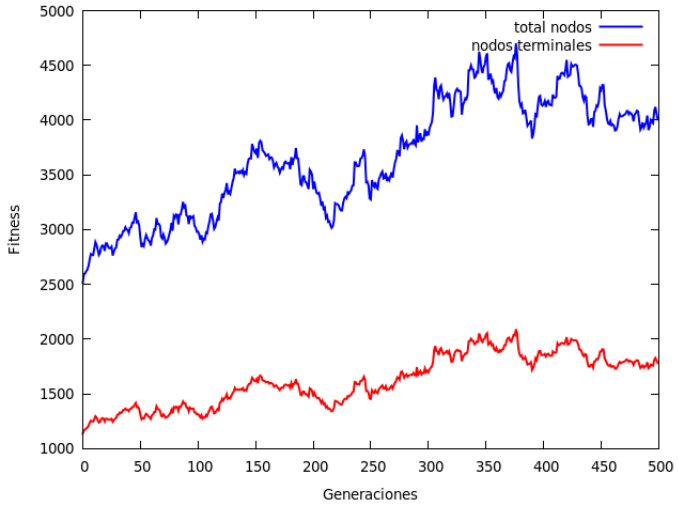

En esta gráfica se nota que es mucho más difícil encontrar una solución al problema mientras más variables hay, en este caso se encuentra una función solución hasta la generación 50 aproximadamente y ahora el rango del fitness del mejor individuo creció entre 0 y 8



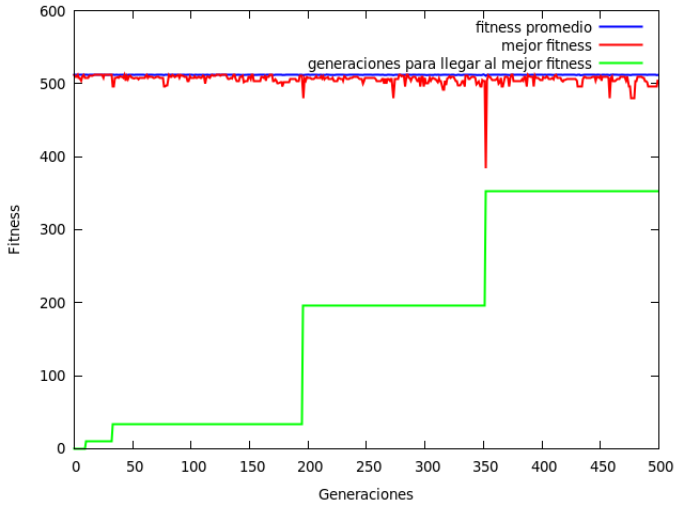
*d*) 7 Variables



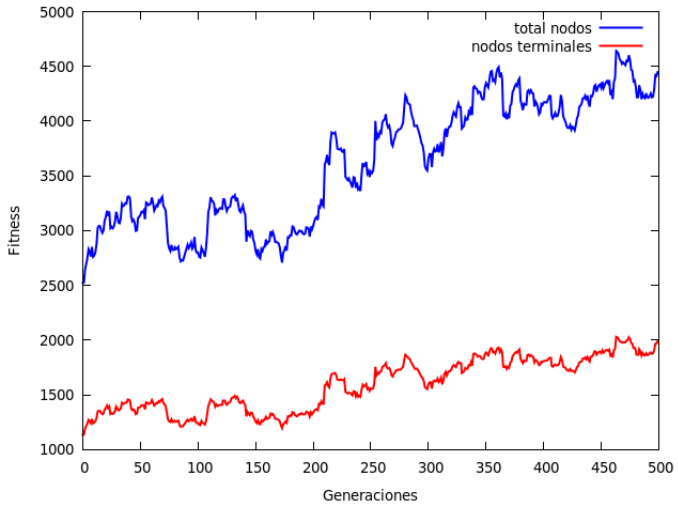

Para 7 variables tenemos una tabla de $2^7 = 128$ renglones, por lo que es más difícil encontrar la función, en este caso no se logró y el mejor individuo tenía un fitness aproximadamente de 40. También podemos ver que conforme el algorítmo avanza, se va encontrando un mejor individuo, antes se encontró rápido y el óptimo, ahora este individuo se mejoró en aproximadamente 5 ocasiones, se puede ver en la línea verde.



*e*) 10 Variables



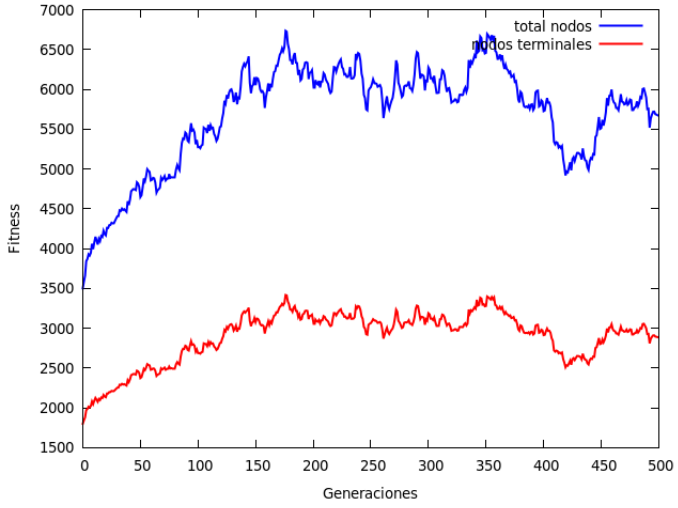

Es un comportamiento muy similar a la gráfica del problema anterior, esta gráfica está muy padre porque se aprecia excelente que cuando la línea roja encuentra un mejor individuo la verde se actualiza a ese valor de la generación. Cabe destacar que el rango del mejor al peor de los mejores fitness es super amplio, es de quizá más de 100 unidades de diferencia.



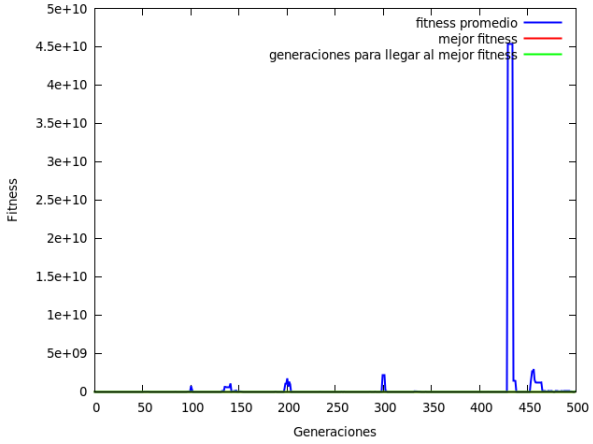
2. SUMA A 10

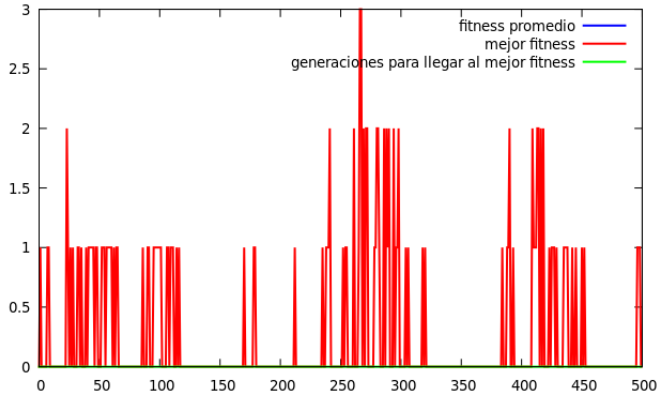
*a*) Con suma


Aquí podemos ver que el espacio de búsqueda es considerablemente más grande que en los problemas anteriores, porque el fitness promedio está un poco más despegado del mejor fitness, tamibién vemos que se encuentran optimos desde el inicio del programa.

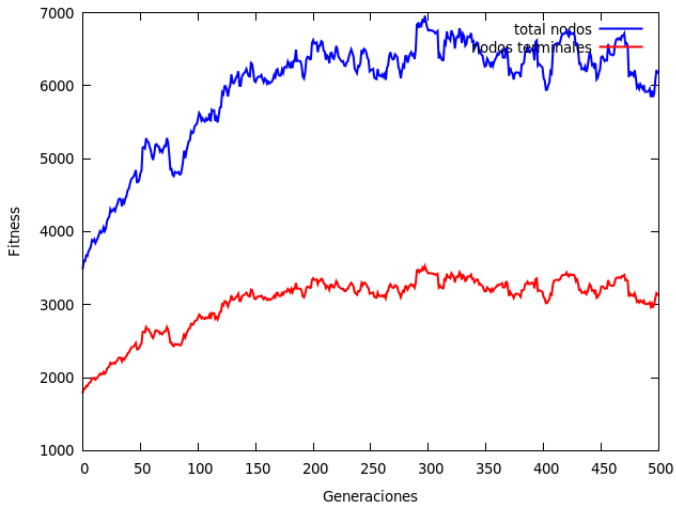


*b*) Con suma y multiplicación

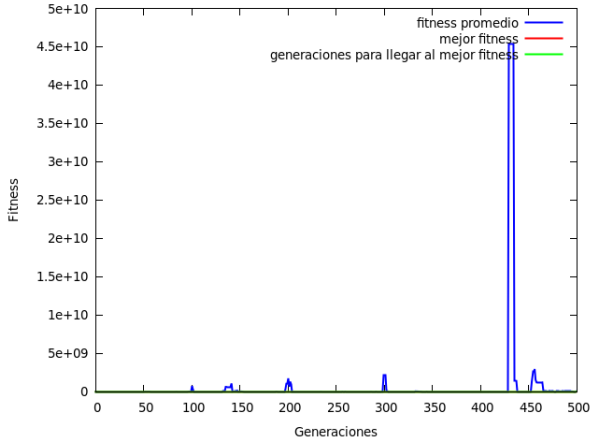


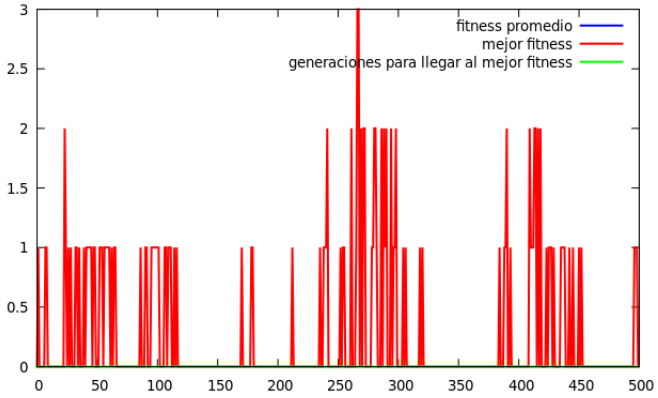




La gráfica anterior es el ejemplo perfecto para mostrar que al aumentar el operador de la multiplicación el espacio de búsqueda crece descontroladamente, en la gráfica de la izquierda no son apreciables las líneas rojas y verdes y en la derecha es un zoom, dejando un poco de lado la línea azul.

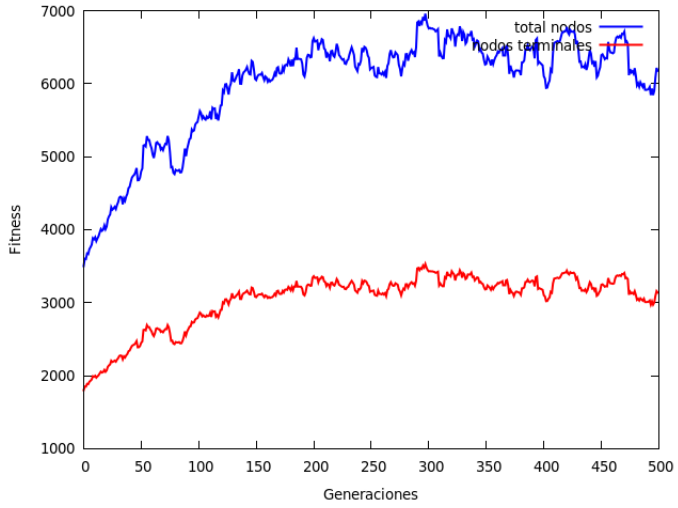


3. Regresión




La situación es exactamente la misma que en la gráfica anterior, cabe destacar que a pesar de amplio espacio de búsqueda se encuentran buenas soluciones con fitness de valor 8.



# 8. Conclusiones

La programación genética nos ayuda al igual que los algorítmos genéticos a explorar grandes espacios de búsqueda, como vimos en clase, las soluciones que encuentran los programas son bastante diferentes a como las haría un ser humano, nada intuitivos y sobre todo mucho más largas, en esta práctica no se pusieron en práctica métodos para acortar estas soluciones, como encontrando identidades aritméticas, reduciendo expresiones, tampoco se tomaron en cuenta propiedades de los operadores tales como la conmutatividad, asociatividad y distributividad, pero para ejemplos más reales todo esto se debe de tomar en cuenta y obviamente vamos a obtener mejores resultados.

Como podemos ver, muchas veces no son suficientes las cosas que tomamos en cuenta para resolver un problema, pues se puede salir de las manos, pues con pequeñas variaciones las complicaciones que se pueden presentar pueden crecer mucho, el ejemplo claro es el de la paridad par para 10 variables, hay mucha diferencia entre las 7 variables y las 10.

También es interesante ver que a pesar de que el fitness promedio es en general muy alto con respecto al mejor fitness por población, eso quiere decir que estamos explorando una muy buena parte del espacio de búsqueda, pues abarca soluciones desde muy buenas hasta unas no tan buenas, para esto cabe destacar que una muy mala solución con un pequeño cambio podría convertirse en un buen candidato, por lo que es importante no menospreciar a los individuos con fitness muy altos.
Por otro lado, también es interesante ver como el afectar el conjunto de operadores puede significar cambios importantes en los fitness logrados, pues en el ejercicio de la suma a 10, el hecho de meter el operador de * cambió muchisimo el fitness promedio, sin embargo se siguieron encontrando buenos resultados, quizá más difícil que en el anterior, pero al meter un nuevo operador, debemos de tomar en cuenta que agregar o quitar un operador cambia muchisimo el espacio de búsqueda y en consecuencia aumenta o disminuye su tamaño (que al final la aplicación de esto es en general en espacios grandes).

La progración genética es una manera muy sencilla de encontrar soluciones a problemas de complejidades elevadas, por ejemplo problemas de la clase de los problemas NP, como seres humanos cuesta trabajo encontrar posibles soluciones, así que a esto ayudaría usar la programación genética y al final utilizamos esos resultados, ya que a fin de cuentas son soluciones considerablemente buenas las que por lo general encontramos.