# Optimización de Constantes Numéricas en Regresión Simbólica utilizando un Framework de Tree-Based Genetic Programming

**Elli, Soledad[1], Jimenez, Victor A.[2], Will, Adrian[1,2], Rodríguez, Sebastián[2]**
[1] *Universidad Nacional de Tucumán, Facultad de Ciencias Exactas y Tecnología*
[2] *Grupo de Investigación en Tecnologías Avanzadas, U.T.N. – F.R.T.*

**Abstract**

*La programación genética (PG) es un conjunto de técnicas de computación evolutiva que permiten resolver problemas automáticamente y que están basadas en Algoritmos Genéticos (AG). La PG ha sido un método muy utilizado para encontrar y evolucionar nuevas e inesperadas maneras de resolver problemas, en particular problemas de Regresión Simbólica (RS). A pesar de esto, uno de los problemas de la PG es la falta de exploración y optimización de las constantes numéricas (o parámetros) dentro de la estructura de árbol con la que se representa un programa. Este trabajo se enfoca en investigar e implementar distintos métodos para la optimización de estas constantes que se encuentran en las terminales de los árboles, de manera de comparar las ventajas relativas de cada método. Los métodos implementados fueron validados con 7 problemas de benchmark.*

**Palabras Clave**
Programación Genética, Tree-Based Genetic Programming, Optimización de constantes numéricas, Regresión Simbólica.

## 1. Introducción

La programación genética es una metodología basada en algoritmos evolutivos, los cuales están inspirados en la evolución biológica. Es una especialización de los algoritmos genéticos que intenta encontrar soluciones a problemas a partir de la inducción de programas. Existen numerosas variantes de estos algoritmos, que se diferencian principalmente por la forma de representar estos programas. En particular, Tree-Based Genetic Programming (Tree-GP) utiliza la codificación basada en estructura de árboles [1].

De manera similar a los AG, Tree-GP inicia con una población cuyos individuos son generados aleatoriamente. A esta población se le aplica operadores de selección, cruzamiento y mutación, lo cual deriva en una nueva población de individuos en cada generación. Este proceso se repite hasta que se verifica una condición de terminación, que por lo general consiste en alcanzar una cierta cantidad de generaciones. Al igual que en la naturaleza, este es un proceso con muchas componentes aleatorias, y no hay hipótesis sobre el problema que permitan garantizar que se encontró el óptimo (como derivadas, convexidad, etc.). En la resolución de problemas con estos métodos se considera suficiente encontrar una buena solución, aunque no sea necesariamente el óptimo global [2], [3], [4].

Una de las aplicaciones de la PG y un área activa de investigación es Regresión Simbólica (RS). Esta técnica consiste en encontrar, a partir de un conjunto de datos experimentales, la expresión algebraica que identifica su comportamiento. Dado que es necesario encontrar una respuesta de manera tangible a datos obtenidos, surge la pregunta (y la propuesta): ¿Cómo evidenciar las relaciones que existen entre una variable dependiente $y$, y una o más variables independientes $(x_1, x_2, \ldots, x_n)$ para poder predecir, pronosticar, conocer o controlar algo? Es ahí cuando entra en juego la PG ya que, debido a la naturaleza de cada problema, no todos se pueden resolver de manera determinística usando métodos numéricos que respondan a los datos de entrada. GP nace para dar respuesta a estos problemas.

En el caso de RS aplicada a GP, la función de Fitness debe estar diseñada para dar mejores valores a soluciones que ajusten mejor los valores de salida sobre los datos de entrada dados. En el presente trabajo la función fitness se define como el Error Cuadrático Medio (ECM):

$$ECM_i = \sqrt{\frac{\sum|valorOriginal_i - valorCalculado_i|^2}{n}}$$

La particularidad que tiene la PG sobre los problemas de RS es que los nodos de los árboles que forman los individuos pueden ser de dos tipos: operador $(+, -, *, /)$ y nodo terminal que, a su vez, puede ser una variable o una constante $(a, y, a, 1.2, 5)$. Si consideramos sólo operadores de aridad dos, podemos representar las expresiones con árboles binarios. Esta simplificación permite evaluar la expresión recorriendo el árbol recursivamente en preorden calculando el valor en cada nodo. En la figura 1 se observa un ejemplo de la representación de una ecuación particular mediante un árbol binario.

En general en PG y como método base (ver [1]), como primera aproximación se fija un conjunto de constantes numéricas que se juzgan oportunas o importantes para el problema (por ejemplo $1, 2, 3, \pi, e$). Sin embargo es muy probable que este conjunto de valores prefijados no sea suficiente para obtener buenas soluciones, por lo que es necesario el uso de algoritmos que exploten y optimicen estas características de las expresiones obtenidas por el AG.

$$f(x) = \frac{sen\big(x(-6.39589 + 3.27152)\big) - \text{sen}(-3.39689)}{1.47056^x}$$

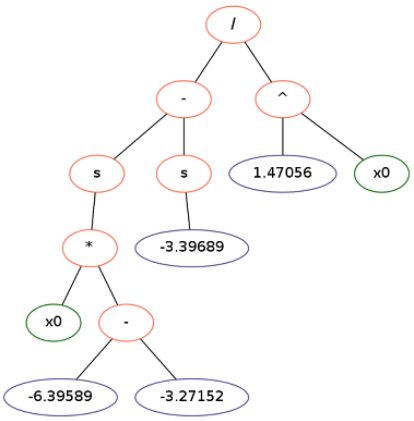


Figura 1: Ejemplo de un árbol de regresión simbólica.

El objetivo del trabajo es implementar los métodos más conocidos para manejo de constantes en Tree-GP, y analizar sus ventajas y desventajas en una serie de problemas conocidos, de manera de poder utilizarlos más eficientemente en problemas reales. Para lograr este objetivo se utilizó y adaptó un framework de Programación Genética desarrollado en trabajos anteriores [5], [11]. Este framework desarrollado en lenguaje C++ cuenta con la estructura básica de los algoritmos genéticos para su aplicación en programación genética basada en Tree-GP. Su diseño permite utilizar diferentes operadores genéticos tradicionales, pero a su vez permite el uso de operadores particulares, que el usuario final puede implementar para problemas específicos.

Este trabajo está estructurado de la siguiente manera: En las siguientes secciones se describen las características generales del problema en particular de la optimización de las constantes en PG para RS. Luego se describen los métodos implementados y utilizados en los casos de estudio y, por último, los resultados obtenidos y conclusiones.

## 2. Problema de las constantes numéricas

La problemática de las constantes tiene su origen en una de las debilidades que presenta la PG, el descubrimiento de los valores numéricos en los nodos terminales. Como se dijo anteriormente, Tree-GP utiliza un conjunto de funciones y un conjunto de terminales en el momento de armar los árboles que representarán a los individuos. En ambos conjuntos se pueden encontrar una variedad muy amplia de elementos en función del dominio del problema que se intenta resolver.

En este punto podríamos formularnos una pregunta muy importante: ¿Por qué la PG es tan vulnerable a problemas de elección de constantes, si los AG de los cuales deriva son especialmente buenos para resolver justamente este problema? Y la

respuesta a esa pregunta tiene que ver con la codificación utilizada en la PG [1]. El problema surge porque los operadores genéticos para Tree-GP operan sobre la estructura de un árbol, y no sobre los valores que están contenidos en los nodos de éste, no permitiendo que estos evolucionen hacia sus valores óptimos.
La forma de afrontar el problema de las constantes numéricas es uno de los factores más importantes para lograr el éxito del algoritmo y mejorar los tiempos de ejecución del mismo. Si la expresión obtenida tiene las constantes apropiadas es más probable que el algoritmo obtenga a una mejor solución que si no las tiene. Por otro lado, mientras más constantes estén involucradas en el problema, mayor será el tiempo necesario para encontrar buenos valores numéricos para las mismas.

## 3. Métodos para optimizar constantes

Existe una gran variedad de métodos para optimizar constantes en RS. De los métodos más simples podemos nombrar:

- Constante aleatoria efímera
- Constantes pre-especificadas
- Funciones sin argumentos
- Combinación Aritmética
- Génesis Aritmético

Además de estos métodos existen otros más complejos que intentan solucionar problemas específicos relacionados a la búsqueda de constantes numéricas. Por lo general, los métodos más sofisticados dependen del fitness y/o de algún parámetro adicional que, combinados de cierta forma permiten obtener un mejor valor para la constante numérica a reemplazar. Algunos métodos definen un rango para limitar los valores numéricos que se obtienen de las modificaciones que sufren las constantes a través de las generaciones o de acuerdo al valor de fitness. A continuación se describen los métodos que se implementaron para el desarrollo de este trabajo.

### 3.1 Numeric Mutation

Numeric Mutation (NM) [6] propone reemplazar cada constante por un valor generado aleatoriamente dentro de un rango determinado. El rango se determina tomando el valor actual de la constante y sumándole o restándole el valor del factor de temperatura actual. El factor de temperatura (*TF*, *Temperature Factor*) está determinado por el producto entre el valor del fitness del mejor individuo de la generación actual ($bestFitness_i$) y una constante especificada por el usuario (*TC*, *Temperature Constant*).

$$TF = bestFitness_i * TC$$

La efectividad del método puede explicarse de la siguiente forma. Cuando el mejor individuo de la población tenga un bajo fitness, el rango de selección será más grande, por lo que el cambio en los valores de las constantes numéricas será mayor. Por el contrario, cuando existan buenos individuos en la población, el valor del fitness se aproximará a cero, lo que provocará que el rango de variación se reduzca progresivamente favoreciendo la búsqueda local alrededor de la mejor solución. Es importante destacar que el funcionamiento se basa en que el Fitness **se aproxime a cero** ($bestfitness_i$ tiende a cero cuando la solución se aproxima a la solución óptima objetivo de la RS).

### 3.2 Hill Climbing

Hill Climbing (HC) es una técnica tradicional de optimización heurística que pertenece a la familia de los métodos búsqueda locales sin uso de derivadas. La relativa simplicidad del algoritmo hace que sea popular para la elección entre los algoritmos de optimización. Es por esto que, a pesar de ser un método de búsqueda local y optimización heurística, se lo aplica con éxito para encontrar y evolucionar constates numéricas en la PG.

### 3.3 Simulated Annealing

Simulated Annealing (SA) es una evolución del método anterior. A diferencia de éste, SA es una heurística de búsqueda global ya que posee mecanismos que le permiten realizar la búsqueda más allá del óptimo local más cercano.

La aceptación de malas soluciones es una propiedad fundamental de este tipo de heurísticas ya que permite salir de la zona de óptimos locales. Es decir, en el caso particular de SA para manejo de constantes en Tree-GP, se adapta el concepto de temperatura y enfriamiento usual de SA, para relacionarlo a la cantidad de generaciones faltantes hasta el final de la corrida y la calidad del individuo encontrado. Así, si el fitness de un individuo es bajo, entonces el factor de temperatura ($T_o$) que controla esta propiedad se incrementa, de forma de abarcar un rango más amplio. Por el contrario, si el fitness del mejor individuo de la generación es mayor, entonces el factor de enfriamiento se reduce, provocando cambios más pequeños. Es así como se toma este concepto para aplicarlo a la generación y aceptación de nuevas constantes en un individuo.

### 3.4 Multi-Dimensional Hill Climbing

Este método recibe su nombre porque utiliza Hill Climbing en cada una de las constantes, en cada dimensión [7]. Multi-Dimensional Hill Climbing o (MDHC) consiste en elegir un número $\Delta$ al azar entre 0 y el factor de temperatura actual, para cada una de las constantes numéricas que se desean modificar. Luego de obtenido $\Delta$, se consideran $k$ nuevos números por cada una de las constantes que se tienen: $c - \Delta$ y $c + \Delta$. Posteriormente se evalúa el árbol correspondiente al individuo en cuestión reemplazando las constantes por los nuevos valores. Se deben probar todas las combinaciones posibles, de las cuales la combinación que produjo un individuo con mejor fitness será la que se conserve.

La principal ventaja de este método es que asegura que la solución obtenida será igual o mejor que la actual en cuanto a Fitness, ya que se evalúan todas las posibles alternativas conservando la mejor.

Sin embargo la implementación de este método requiere mucho procesamiento ya que la cantidad de evaluaciones de la función fitness se incrementa exponencialmente con la cantidad de constantes involucradas. La cantidad de evaluaciones adicionales requerida en cada generación es:

$$n = 3^{Cant\_Constantes} * Cant\_Individuos$$

Se debe considerar que la evaluación suele ser la operación que consume más procesamiento en un algoritmo genético, por lo tanto los tiempos de ejecución pueden incrementarse considerablemente.

### 3.5 Differential Evolution

Differential Evolution (DE) es un método que optimiza un problema iterativamente tratando de mejorar una solución candidata con respecto a un determinado fitness [8]. Una de las variantes más sencillas del algoritmo DE trabaja en base a una población de soluciones candidatas, llamados agentes. Estos agentes se mueven alrededor en el espacio de búsqueda mediante la elección arbitraria de tres individuos distintos al azar de la población de constantes $X_1, X_2, X_3$ y un lugar de asignación $i$. Entonces:

$$X_i = X_1 + F * (X_2 - X_3)$$

dónde $F$ es básicamente un factor de escala.

Algunas variantes de DE se diferencian en la cantidad de puntos considerados, otras toman $X_1$ (el vector de constantes numéricas) del mejor o los mejores individuos de la población.

DE es un método sencillo y rápido, pero presenta una pobre estabilidad a repeticiones debido a que tiene una componente aleatoria muy fuerte.

## 4. Pruebas

En esta sección se presentan las pruebas realizadas con los algoritmos implementados.

### 4.1 Testeo de métodos

La función de *Rastriguin* es una función diferenciable y no convexa, multimodal, comúnmente utilizada como benchmark para probar la eficiencia y rendimiento de los algoritmos de optimización. Como el problema planteado radica en la obtención de constantes dentro de un problema de RS y no en la optimización global de un problema, se usó esta función con sus constantes modificadas y definida para una sola variable independiente de manera que permita probar la eficacia y performance del sistema en casos difíciles. Teniendo en cuenta esto, la ecuación final para ser evaluada es:

$$f(x) = 0.95x^2 - 5.5\cos(\pi x)$$

Los parámetros utilizados para las pruebas, dependiendo de cada método, fueron:

- Generaciones: 500
- Tamaño Población: 500
- Cruzamiento: 80%
- Mutación: 10%
- Elite: 5%
- Set de Funciones: +, -, *, /, *sen*, *cos*
- Porcentaje de población afectada a la optimización: 80%
- TC: 0.002
- Iteraciones (HC, SA): 15
- Tamaño del paso Máximo (HC, SA): 1
- Temperatura inicial (SA): 2
- Constante de enfriamiento (SA): 0.05
- Rango de definición: [-5, 5]
- Generaciones Máximas: 10
- Altura máxima del árbol: 6

#### 4.1.1 Numeric Mutation

Utilizando los parámetros de entrada descritos anteriormente, la mejor solución obtenida con este método se muestra en la figura 2. Luego de un post-procesamiento, la ecuación final corresponde a:

$$f(x) = \cos(3.13881x) * 0.950526x^2 - 0.422558\cos(3.13993\,x)$$

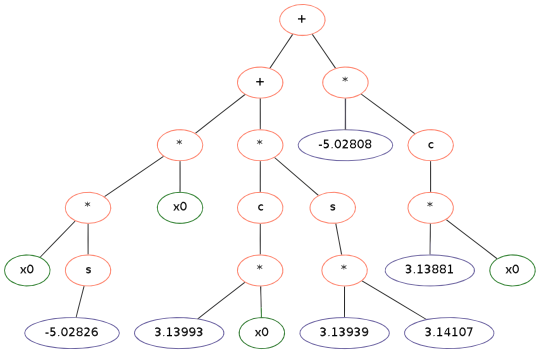


Figura 2: Mejor individuo obtenido con NM.

El fitness que obtuvo el mejor individuo de esta prueba fue igual a $-0.0649662$ en un tiempo total de 146,21 segundos. En la figura 3 se puede ver la curva correspondiente al mejor individuo de salida comparado con la función original.

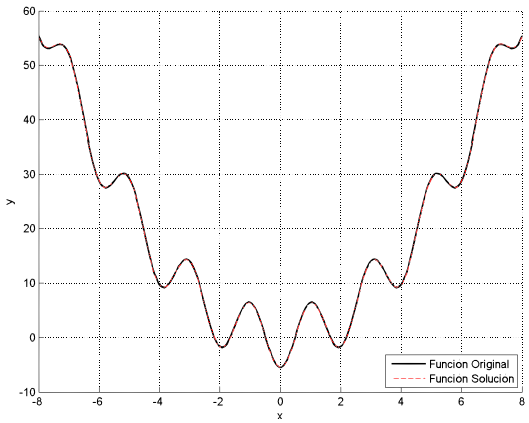


Figura 3: Función de salida vs función original para NM.

#### 4.1.2 Hill Climbing

Con este método se obtuvo un individuo final correspondiente al de la figura 4, y una ecuación simplificada equivalente a:

$$f(x) = -0.3432x^2(\cos(3.2277\,x) - 3.4554)\cos(0.8047\cos(1.4139x))$$

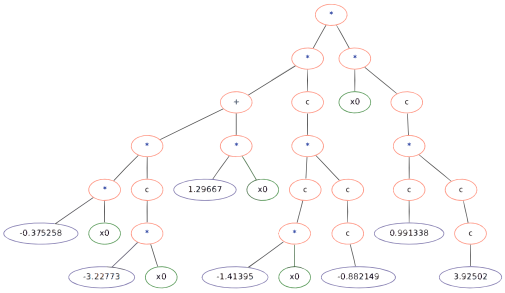


Figura 4: Mejor individuo obtenido con HC.

La ejecución se llevó a cabo en 8337.71 segundos y se obtuvo un fitness igual a $-2.28326$. La solución obtenida y el desempeño de éste método para el ejemplo planteado se puede ver en la figura 5. Si bien se observa que el algoritmo sigue la tendencia general de la función, los resultados no son tan buenos como en los otros casos.

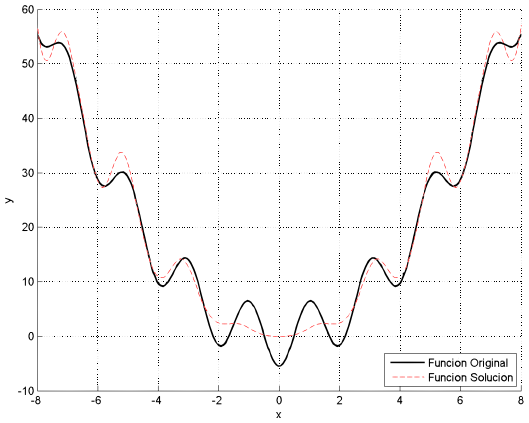


Figura 5: Función de salida vs función original para HC.

### 4.1.3 Simulated Annealing

Con este método se obtuvo el individuo representado en la figura 6 y como salida la ecuación simplificada:

$$f(x) = x^2 - sen(-x + sen(x) + 0.5309) - 5.0079\cos(3.1382x) - 0.9909$$

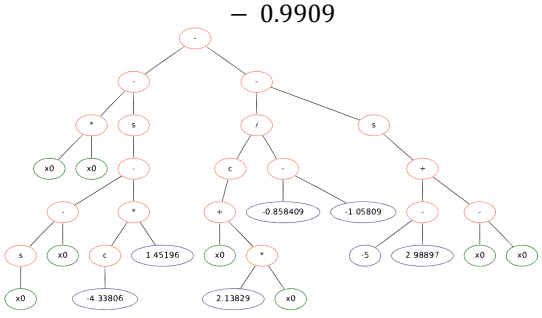


Figura 6: Mejor individuo obtenido con SA.

Diferente al caso anterior, este método pudo aproximar en su mayoría los datos de entrada, logrando un fitness de -1.09076 en 11150.8 segundos, como se puede ver en la figura 7.

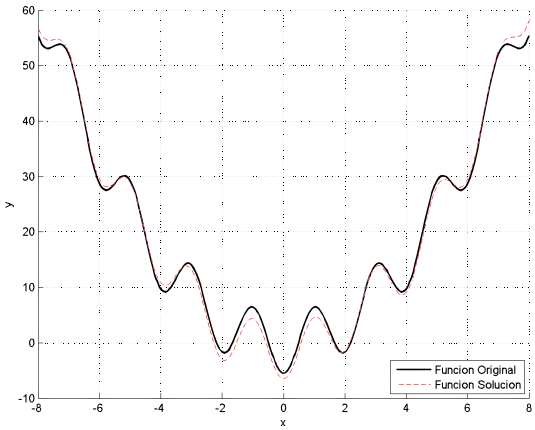


Figura 7: Función de salida vs función original para SA.

### 4.1.4 Differential Evolution

Con este método se obtuvo el individuo de la figura 8 y como salida la ecuación simplificada:

$$f(x) = -sen(1.5803 - 3x) + x^2 - sen(3x + 8.2301)$$

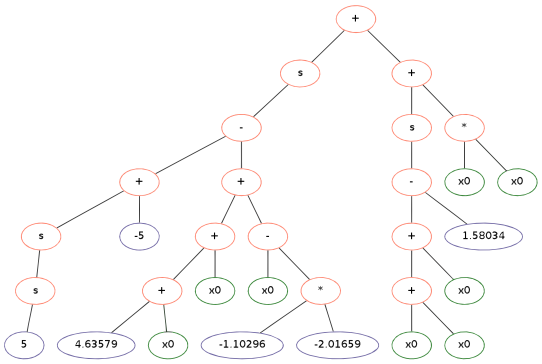


Figura 8: Mejor individuo obtenido con DE.

A partir de la figura 9 y el valor de fitness obtenido se concluye que es necesaria una segunda corrida para obtener mejores resultados y reutilizar el resultado previo encontrado por el sistema, al igual que en el caso de HC.

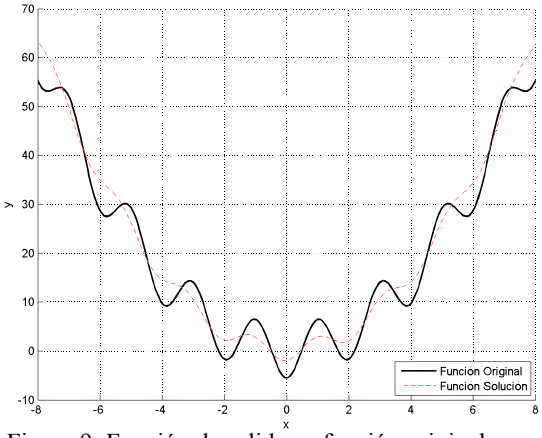


Figura 9: Función de salida vs función original para DE.

### 4.1.5 Multi Dimensional Hill Climbing

Con este método se obtuvo el individuo de la figura 10 y como salida la ecuación simplificada:

$$f(x) = 0.9519x^2 - 5.5048\cos(3.1391x)$$

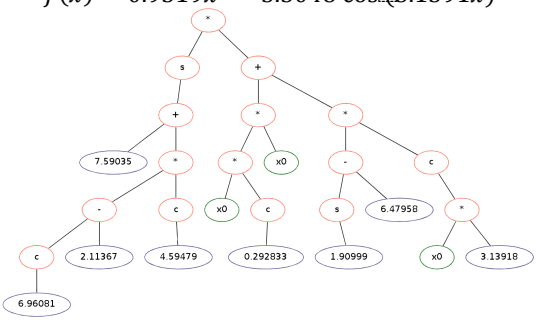


Figura 10: Mejor individuo obtenido con MDHC.

El individuo consiguió un valor de fitness de $-0.0778178$ en $1506.56$ segundos como se puede observar en la figura 11.

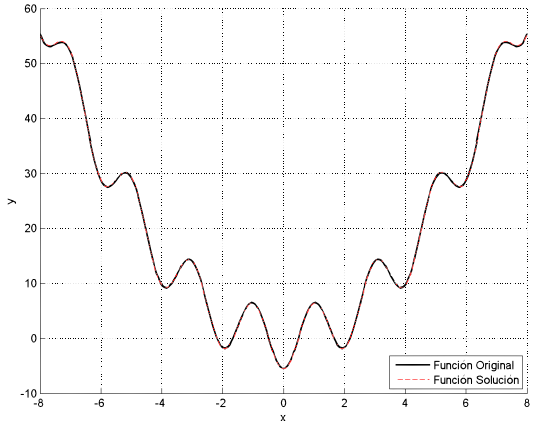


Figura 11: Función de salida vs función original para MDHC.

## 4.2 McCormick Function

*McCormick* es otra función utilizada como benchmark para problemas de optimización [9]. La misma está definida por la siguiente ecuación y representada por la gráfica de la figura 12.

$$f(x,y) = sen\,(x+y) + (x-y)^2 - 1.5x + 2.5y + 1$$

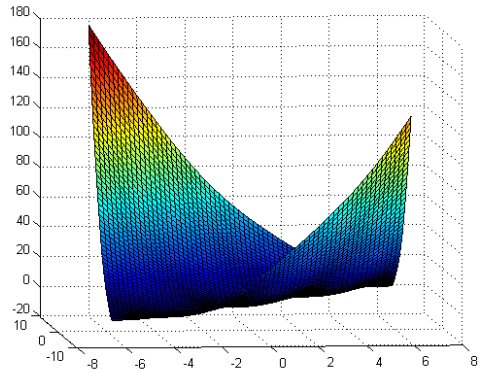

Figura 12: Gráfica de McCormick Function

Esta función de benchmarking fue probada con el algoritmo MDHC, el cual demostró tener una performance superior en comparación con los otros métodos ante problemas más avanzados. Los parámetros de ejecución del sistema fueron:

- Generaciones: 200
- Tamaño de la población: 150
- Cruzamiento: 80%
- Mutación: 30%
- Elite: 5%
- TC: 1.6
- Set de Funciones: +, -, *, /, *sen*, *cos*

El individuo de salida corresponde al árbol de la figura 13 y su ecuación simplificada es:

$$f(x) = x^2 - 2xy - 1.3323x + y^2 + 2.6209y + 1.0887$$

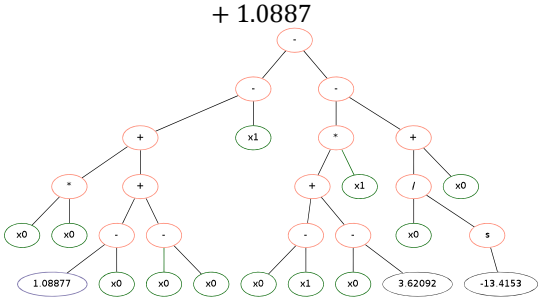

Figura 13: Mejor individuo obtenido.

Este individuo consiguió un fitness igual a $-1.40393$ en $7980.15$ segundos de ejecución. El algoritmo encontró fácilmente las componentes principales debido a que sus valores numéricos son muy significativos comparado con los otros términos. Por este motivo no fueron encontradas las componentes más pequeñas. Para solucionar esto se obtuvo un residuo, calculando la diferencia entre los resultados obtenidos por el algoritmo en la primera corrida con los datos de prueba. Luego se hizo una segunda corrida con el residuo calculado, utilizando los mismos parámetros y llegando a los siguientes resultados:

$$f(x) = 0.3999sen(x+y) - 0.1999x - 0.1999y - 0.1998$$

Lo que nos lleva a la ecuación final del sistema:

$$f(x) = x^2 - 2xy + 0.3399sen(x+y) - 1.5322x + y^2 + 2.4209y + 0.8889$$

Esta ecuación posee todos los términos de la función original. La diferencia entre las constantes encontradas es menor a $0.15$, excepto por el coeficiente del la función seno que está un poco más alejado ($0.66$). Esto se debe a que en el intervalo considerado, la función seno tiene un valor máximo en valor absoluto de 1, que es 2 órdenes de magnitud menor que los de mayor grado. Para mejorar este resultado se requieren métodos de ajuste local como regresión lineal, o métodos con derivada si se sospecha que hay constantes dentro de la función seno (en lugar de sólo $x + y$). Es importante destacar que incluso en la primera fase descripta, se obtuvo un fitness de $-1.4$, lo cual indica que la función está próxima en valor absoluto a la original. El error obtenido en el individuo final se puede ver en la figura 14.

Para este caso de prueba se requirió una segunda corrida del sistema teniendo en cuenta las componentes no encontradas en la primera solución para poder llegar a un resultado razonablemente bueno.

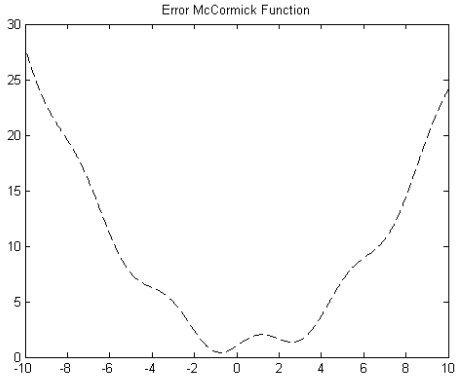


Figura 14: Gráfica del error obtenido en la solución final.

### 4.3 Pasajeros de Aerolínea Internacional

Esta es una prueba con datos reales publicado para su reproducción y análisis [10]. El caso consiste en una serie de tiempo, en donde se involucran la cantidad de pasajeros de una aerolínea internacional (medidos en cientos de pasajeros) y la fecha de viaje, tomando como referencia el periodo de Enero de 1949 a Diciembre de 1960.

La prueba se realizó con Multi-Dimensional Hill Climbing utilizando los siguientes parámetros:

- Población: 180
- Generaciones: 450
- Cruzamiento: 65%
- Mutación: 10%
- Elite: 5%
- TC: 1.
- Set de Funciones: +, -, *, /, *sen*, *cos*, *pow*, *sqrt*.

En la figura 15 se pueden observar los resultados obtenidos y comparar la semejanza entre la serie de tiempo original (línea llena) y la obtenida con el framework (línea punteada).

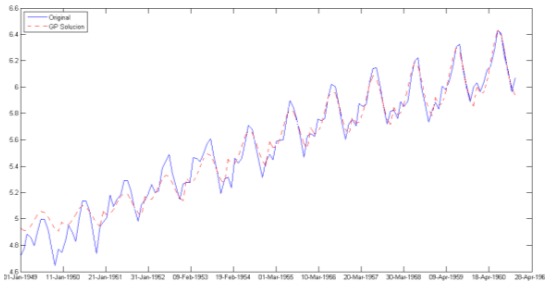

Figura 15: Función de salida vs función original.

A sí mismo, en la figura 16 se puede observar el árbol de salida con sus constantes correspondientes. Al recorrer el árbol en preorden se puede obtener la función que define a esta serie de tiempo.
Esta reproducción obtuvo un fitness igual a $-19.9834$ en $250.48$ segundos, resultados en los cuales se ve reflejada la complejidad del problema. El análisis realizado en el trabajo original muestra que la serie tiene componentes autorregresivos (se debe incorporar datos de fechas anteriores como variables de entrada), lo cual mejoraría los resultados. Sin embargo se nota el potencial de la herramienta consiguiendo el resultado indicado.

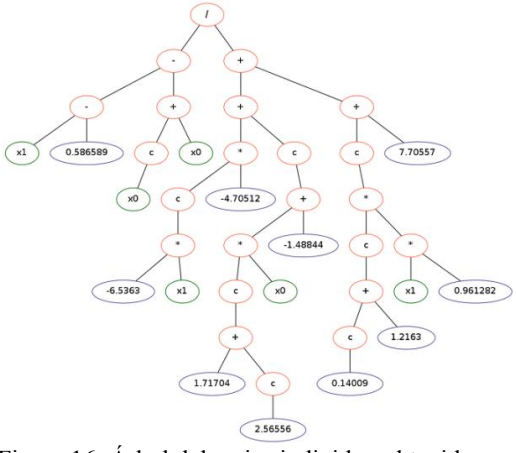


Figura 16: Árbol del mejor individuo obtenido.

## 5. Resultados y Discusión

En las pruebas realizadas a funciones de benchmarking el sistema responde correctamente a los datos de entrada. En la mayoría de los casos se obtuvieron las componentes de mayor variación o de mayor impacto sin mayores inconvenientes, sin embargo las componentes pequeñas (residuo) no siempre fueron encontradas.
Como se puede ver en los resultados obtenidos en el testeo de los métodos, con NM, MDHC y SA las soluciones encontradas son bastante buenas y aproximadas a los datos de entrada, pero existe una diferencia significativa con los tiempos de ejecución y los requerimientos de procesador y memoria. Al tener que realizar tantas evaluaciones en los métodos generacionales (SA, HC y MDHC) el tiempo necesario para llegar a una buena solución se eleva demasiado y resulta poco eficiente y práctico la utilización de estos métodos para este tipo de problemas. Cabe destacar que NM es mucho más simple y eficiente que los otros cuatro métodos en la mayoría de los casos simples, pero no así para los casos más difíciles.
El Framework utilizado demostró tener la robustez necesaria para la aplicación de problemas de Regresión Simbólica de distintas características obteniendo soluciones de buena calidad, rápida convergencia y tiempos acotados.
Esta implementación abre las puertas a un gran conjunto de problemas que en la actualidad no tienen solución determinística, y en donde dicha solución puede representar una gran mejora en campos de aplicación de cualquier área cuyo funcionamiento sea en base a conjuntos de datos.
Resulta importante destacar que existen otras heurísticas como Redes Neuronales que son capaces de reproducir eficazmente la forma de una función, pero no son susceptibles de análisis posteriores. Las soluciones proporcionadas por PG son fórmulas matemáticas y como tales susceptibles de un análisis teórico posterior, como análisis de sensibilidad y de propagación de errores, e incluso dar lugar a la generación de mejores modelos teóricos sobre la base de las fórmulas encontradas.

## 6. Conclusiones

De lo analizado y probado se concluye que la herramienta sólo está destinada a ser una ayuda y una aproximación a una solución analítica, siendo necesario en todos los casos un procesamiento matemático posterior y un análisis por parte de expertos para conseguir resultados utilizables.

Dicho esto, resulta importante destacar que el problema demostró ser muy complejo ya que existen distintos métodos de resolución para los distintos problemas que se pueden presentar.

Las aplicaciones para el sistema desarrollado superaron las expectativas tanto en tiempo como en performance. En las pruebas efectuadas probó ser estable a repeticiones en la mayoría de los casos, encontrando una solución muy aproximada en cada problema planteado. Esta es una de las características más difíciles de lograr debido a la aleatoriedad intrínseca de los algoritmos genéticos de los que se basa la RS dentro de un sistema de PG. Específicamente se puede resaltar los siguientes puntos:

- Se lograron excelentes resultados en los casos de benchmark y en los problemas comparados de la bibliografía. Los tiempos de ejecución son estables y proporcionales a la complejidad del problema.
- En los casos más complejos, combinando más de 6 términos, con potencias y otros tipos de funciones combinadas, se observa una disminución en la tasa de éxitos.
- El programa encuentra la respuesta correcta en 1 de cada 5 veces aproximadamente, y en varios casos es necesario realizar el proceso en varias etapas, haciendo un análisis por residuos para encontrar un resultado correcto.
- Los métodos anteriormente mencionados en al comienzo de la sección 2 son sumamente básicos para un problema tan complejo, y fueron descartados en etapas preliminares del sistema.
- Se pudo concluir que el método de NM es sumamente rápido y eficiente para casos relativamente simples, pero no asegura buenos resultados en todos los casos.
- DE es eficiente en procesamiento pero tampoco asegura resultados, siendo sobrepasado en eficiencia y resultados por NM.
- Los métodos de SA y HC proporcionan buenos resultados pero requieren mucho más procesamiento y ajustes experimentales que el resto de los métodos.
- El método MDHC es sumamente costoso para grandes cantidades de constantes, pero aplicarlo con un valor de $n$ relativamente bajo (como $n$ = 3 o 4) asegura que esas constantes tomarán el mejor valor posible, lo que mejora los resultados sin aumentar exageradamente la cantidad de procesamiento.
- En la codificación actual del sistema se aplica un método por vez. A futuro claramente es deseable aplicar más de un método en forma simultánea de forma de aprovechar las ventajas de cada uno.
- Para los casos de prueba se tuvieron en cuenta un grupo mínimo de constantes numéricas de entrada, por lo general en un rango [-5,5], sólo para que el sistema sea capaz de introducir en las primeras generaciones estas constantes para que luego sean optimizadas por los algoritmos. Todas las pruebas se corrieron en una máquina virtual bajo un entorno de Linux con un solo procesador Core i5 de 2.67 Mhz y 3GB de memoria RAM.
- La herramienta desarrollada y la PG a nuestro juicio tiene como objetivo principal proponer un modelo que debe ser analizado y mejorado con herramientas de búsqueda local para mejorar las constantes conociendo la estructura de la solución, desde regresión lineal, métodos con derivadas, hasta métodos estadísticos de selección de parámetros o específicos para series de Fourier, que permitan obtener los valores correctos dado que se conoce la estructura de la solución.

## 7. Trabajos Futuros

Como parte del trabajo futuro correspondiente a la investigación presentada, se tienen en cuenta las siguientes observaciones:

- Resulta imprescindible mejorar la definición de convergencia correcta. En algunos casos se obtienen fitness excelentes de 0.01 o menores, pero sin embargo en el gráfico correspondiente se observa que el ajuste no es perfecto. Sin embargo, esto no se considera un problema en ciertos casos en donde el objetivo es solamente detectar tendencias generales en los datos (como datos económicos o bursátiles).
- Mejoras en la metodología para analizar los residuos de las soluciones obtenidas en los casos en que el fitness no se acerca a los valores deseados, siendo necesario correr nuevamente el algoritmo sobre la diferencia de los datos con los primeros resultados (residuo).
- Implementar mejoras para encontrar soluciones más simples a un problema. Esto puede lograrse con un sistema para eliminación de intrones o simplificación de fórmulas.

**Referencias**

[1] Poli, R., Langdon, W. B., McPhee, N. F., y Koza, J. R., "A field guide to genetic programming", *Lulu .com*, 2008. URL: http://books.google.com.ar/books?id=3PBrqNK5fFQC, accedido el 24/08/2014

[2] Goldberg, D. y Richardson, J., "Genetic algorithms with sharing for multi-modal function optimization", en *Genetic Algorithms and Their Applications: Proceedings of the Second International Conference on Genetic Algorithms*, pp. 44 - 49. ICGA, 1987.

[3] Borenstein, Y. y Poli, R., "Empirical approach for theory of randomized search heuristics", 2006.

[4] Xu, X. y Gen, M., "Introduction to Evolutionary Algorithms. Decision Engineering", *Springer*, ISBN 9781849961288, 2010. URL: http://books.google.com.ar/books?id=rHQf_2Dx2ucC.

[5] Santochi, D., Lenis J., "Plataforma de Programación Genética en C++". *Tesis de grado de la carrera de Ingeniería en Computación de la Universidad Nacional de Tucumán*, 2011.[1]

[6] Evett, M. y Fernandez, T., "Numeric mutation: Improved search in genetic programming", en *Diane J. Cook, ed., FLAIRS Conference*, pp. 106-109, AAAI Press, ISBN 1-57735-051-0, 1998. URL: http://dblp.uni-trier.de/db/conf/flairs/flairs1998.html#EvettF98

[7] Fernandez, T., "Evolution of numeric constants in genetic programming", 1997. URL: http://www.cse.fau.edu/~thomas/AI/GP_and_Numeric_Constants.PDF, accedido el 24/08/2014.

[8] Rainer, S. y Kenneth, P., "Differential evolution a simple and efficient heuristic for global optimization over continuous spaces". *Journal of Global Optimization 11*: pp. 341–359, Kluwer Academic Publishers, 1997.

[9] Hyndman, R. J., "Time series data library", 2005. URL: http://data.is/TSDLdemo, accedido el 24/08/2014.

[10] Tang, K., Yao, X., Suganthan, P. N., MacNish, C., Chen, Y. P., Chen, C. M., y Yang, Z., "Benchmark functions for the cec2008 special session and competition on large scale global optimization", 2008. URL: http://nical.ustc.edu.cn/cec08ss.php

[11] Navarro, F., Remis, L., Santochi, D., Lenis, J., Will, A., y Rodriguez, S., "Programación Genética y aplicaciones a Robótica: Cortadora de Pasto y Resolución de Laberintos," en *Actas de las VII Jornadas Argentinas de Robótica, Olavarria*, 2012.

[1] **El informe y el código están disponibles sobre pedido en la biblioteca de la Facultad de Ciencias Exactas y Tecnología de la Universidad Nacional de Tucumán.**