Optimización de Constantes Numéricas en Tree-Based Genetic Programming

Adrian Jimenez[1], Soledad Elli[2], Adrian Will[3], Sebastián Rodríguez[4]
[1,3,4]Grupo de Investigación en Tecnologías Avanzadas (GITIA),
Universidad Tecnológica Nacional – Facultad Regional Tucumán.
[2]Universidad Nacional de Tucumán, Facultad de Ciencias Exactas y Tecnología
[1]adrian.jimenez@gitia.org

Resumen: *La Programación Genética (PG) es un conjunto de técnicas de computación evolutiva basadas en Algoritmos Genéticos, que permiten resolver problemas mediante la generación automática de programas. La PG ha demostrado ser un método eficiente para encontrar soluciones a una gran variedad de problemas donde se cuenta con una función objetivo o tarea a realizar. Sin embargo, una de las principales dificultades que presenta es la exploración y optimización de constantes numéricas (o parámetros). Este trabajo se enfoca en la investigación e implementación de distintos métodos para la optimización de dichas constantes, utilizando un framework de Tree-GP. Se eligió la Regresión Simbólica como aplicación debido a la clara necesidad de encontrar constantes precisas. Los métodos fueron probados en un conjunto de benchmarks, y se determinó que la herramienta logra buenos resultados, pero a medida que crece la complejidad del problema la tasa de éxitos disminuye y el costo computacional se incrementa considerablemente.*

Palabras Claves*: IPrograma ción Genética, Tree-Based GP, Optimización de constantes numéricas, Regresión Simbólica.*

Abstract*: Genetic Programming (GP) is a set of evolutionary computation techniques based on genetic algorithms, which solve problems by automatic generation of programs. The PG has proved to be an efficient method to find solutions to a wide variety of problems that have an objective function or task to perform. However, one of the main difficulties is the exploration and optimization of numerical constants (or parameters). This work focuses on the research and implementation of various methods for optimizing these constants, using a framework of Tree-GP. Symbolic Regression was selected as application due to the clear need for precise constants. The methods were tested on a benchmark set, and we determine that the tool achieved good results, but as the complexity of the problem increases the success rate and decreases the computational cost increases considerably.*

Keywords: *Genetic Programming, Tree-Based GP, Numeric constants optimization, Symbolic Regression.*

INTRODUCCIÓN

La Programación Genética (PG) es un conjunto de técnicas de computación basada en algoritmos evolutivos, en particular en Algoritmos Genéticos (AG). La PG es un conjunto de herramientas que intenta encontrar soluciones automáticamente sin requerir que el usuario conozca o especifique la forma o estructura de la solución a encontrar (Koza, 1998). Esto se logra mediante la generación y evolución de programas computacionales. Existen numerosas variantes de estos algoritmos, que se diferencian principalmente por la forma de representar dichos programas. En particular, Tree-Based Genetic Programming (Tree-GP) utiliza la codificación basada en estructura de árboles (Poli et al., 2008).

Una de las aplicaciones de la PG y un área activa de investigación es la Regresión Simbólica (RS). Esta técnica permite encontrar, a partir de un conjunto

de datos experimentales, la expresión algebraica que identifica los describe. Existen otras heurísticas como Redes Neuronales que son capaces de reproducir eficazmente la forma de una función, pero no producen un modelo matemático utilizable. Las soluciones proporcionadas por PG son fórmulas matemáticas, y como tales, son susceptibles a 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.

Una de las debilidades que presenta la PG para resolver este tipo de problemas es la falta de exploración y optimización de las constantes numéricas (o parámetros) de las expresiones que se obtienen, principalmente debido al tipo de codificación utilizada en Tree-GP. El objetivo de este trabajo es implementar métodos conocidos para búsqueda y optimización de constantes en Regresión Simbólica utilizando 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 (Santochi and Lenis, 2011; Santochi et al., 2012). Este framework desarrollado en lenguaje C++ cuenta con una estructura básica de 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 implementar y utilizar nuevas arquitecturas de Algoritmos Genéticos y operadores que puedan ser necesarios para aplicaciones específicas.

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

REGRESIÓN SIMBÓLICA CON TREE-BASED GP

De manera similar a los AG, Tree-GP es un algoritmo evolutivo, el cual inicia con una población generada aleatoriamente, a la que se le aplican operadores de selección, cruzamiento y mutación derivando en una nueva población. Este proceso se repite en cada generación hasta que se verifica una condición de terminación, que por lo general consiste en alcanzar una cierta cantidad de generaciones o un nivel de error entre la función original y la evolucionada. Al igual que en la naturaleza, este proceso tiene muchas componentes aleatorias, y no hay hipótesis sobre el problema (como derivadas, convexidad, etc.) que permitan garantizar que las soluciones encontradas sean las óptimas. En la resolución de problemas con estos métodos se considera suficiente encontrar una buena solución, aunque no sea necesariamente el óptimo global (Goldberg and Richardson, 1987; Borenstein and Poli, 2006; Xu and Gen, 2010).

CODIFICACIÓN

La particularidad que tiene Tree-GP aplicado a problemas de RS es que las expresiones o soluciones obtenidas son representadas mediante estructuras de árboles. En el caso de expresiones aritméticas, estos árboles se componen de nodos que pueden ser de dos tipos: operadores (por ejemplo +,-,* ,/) y nodos terminales, que a su vez pueden corresponder a una variable o una constante numérica (por ejemplo a,y,a,1.2,5). Si consideramos sólo operadores de aridad uno y dos (es decir operadores con dos parámetros como máximo), las expresiones pueden ser representadas mediante árboles binarios. Esta simplificación permite evaluar

la expresión recorriendo el árbol recursivamente en preorden calculando el valor en cada nodo.

GENERACIÓN DE INDIVIDUOS

Cada individuo de la población inicial se genera tomando aleatoriamente elementos de un conjunto de operadores y un conjunto de terminales prefijados. En general en PG y como método base (ver Poli et al., 2009), los operadores y constantes numéricas incluidas en estos conjuntos son aquellos que se juzgan oportunas o importantes para el problema (por ejemplo, las constantes comúnmente utilizadas son 1,2,3,π,e). Sin embargo es muy probable que este conjunto de valores prefijados no sea suficiente para obtener buenas soluciones. Por este motivo se utilizan otros algoritmos dedicados a explotar y optimizar estas características de las expresiones obtenidas por el AG. Estos algoritmos se describen en las secciones posteriores.

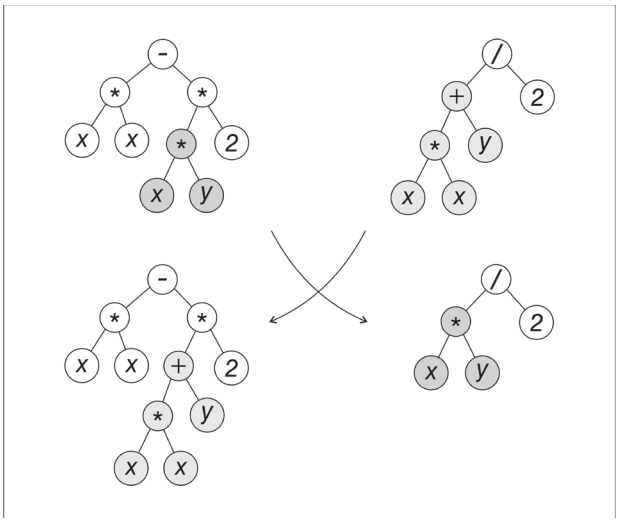

Fig. 1 – Ejemplo de aplicación del operador Tree Crossover.

CRUZAMIENTO

En este trabajo se utilizó Tree-Crossover por ser uno de los operadores de cruzamiento más conocidos y utilizados para Tree-GP. Primero se selecciona un nodo al azar de cada padre. Los hijos se generan a partir de los padres, intercambiando los sub-árboles cuyas raíces son los nodos seleccionados, tal como se muestra en la Fig. 1.

MUTACIÓN

El operador de mutación que se utilizó es la mutación de un punto, donde se selecciona un nodo al azar del individuo y se lo reemplaza junto a todos sus hijos por un sub-árbol generado aleatoriamente. En la Fig. 2 se muestra un ejemplo de aplicación de este operador.

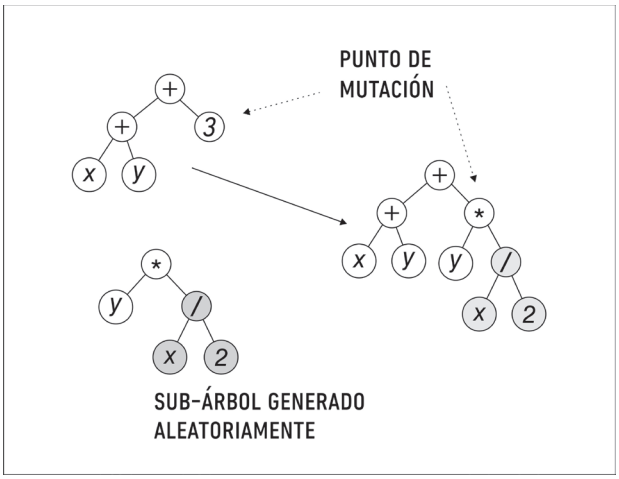

Fig. 2 – Ejemplo de aplicación del operador Tree Mutation.

FUNCIÓN FITNESS

El objetivo de la PG en el caso de Regresión simbólica es encontrar una expresión algebraica que mejor describa un conjunto de datos. Por lo tanto, la función Fitness debe estar diseñada de tal manera que asigne mejores valores a aquellas soluciones que permitan un mejor ajuste de los datos de la variable de salida. Para el desarrollo de este trabajo la función Fitness se calcula mediante la evaluación de los datos experimentales, cuyo resultado son comparados con la respuesta esperada. El valor de fitness será entonces el valor de error obtenido.

La métrica de error utilizada es Root Mean Squared Error (RMSE) definida por la Ec. 1, dónde y_i son los valores reales de la variable de salida y p_i son los valores calculados con la solución obtenida. La ventaja que presenta el uso de esta métrica de error es que se expresa en la misma unidad que la variable de salida, lo que permite su fácil interpretación.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - p_i)^2} \qquad (1)$$

PROBLEMA DE LAS CONSTANTES NUMÉRICAS

Como se dijo anteriormente, Tree-GP utiliza un conjunto de funciones y un conjunto de terminales en el momento de generar los árboles que representarán a los individuos. La problemática de las constantes tiene su origen en una de las debilidades que presenta la PG, el descubrimiento de los valores numéricos del conjunto de terminales. Este problema que presentan los AG para encontrar los valores correctos de las constantes está relacionado a la codificación que utiliza la PG. El mismo surge porque los operadores genéticos (cruzamiento y mutación) 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. Puede ocurrir que se encuentre la expresión correcta con las constantes inapropiadas, y por este motivo puede ser descartada, con lo cual nunca se sabrá si el problema se encuentra en la estructura de la expresión o en los valores de las constantes. Es importante entonces que la expresión obtenida tenga las constantes apropiadas, ya que es más probable que el algoritmo obtenga a una mejor solución que si no las tiene.

MÉTODOS PARA OPTIMIZAR CONSTANTES

Existe una gran variedad de métodos para optimizar constantes en RS (Fernandez, 1997; Koza, 1998). De los métodos más simples podemos nombrar:

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

Estos métodos son sumamente básicos para un problema tan complejo como lo es la optimización de constantes, por lo que en general no dan buen resultado. Por este motivo existen otros métodos 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 forma adecuada, permiten obtener un mejor valor para la constante. 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. El rango elegido está directamente relacionado con la complejidad del problema. Mientras más grande sea éste, aumenta exponencialmente la dificultad del problema (Byrne et al., 2009). A continuación se describen algunos de ellos, utilizados para el desarrollo de este trabajo.

NUMERIC MUTATION (NM)

Numeric Mutation (Evett and Fernandez, 1998) es una técnica utilizada para la creación de nuevas y mejores constantes numéricas durante la ejecución

de PG. Se considera a NM como un operador del algoritmo, como lo son el cruzamiento y la mutación. NM propone reemplazar cada constante por un valor generado aleatoriamente dentro de un rango determinado. Este rango se forma tomando el valor actual de la constante, sumándole o restándole un valor llamado factor de temperatura (T_F, Temperature Factor). El factor de temperatura (Ec. 2) está definido por el producto entre el valor del fitness del mejor individuo de la generación actual (f_best) y una constante (T_C, Temperature Constant) especificada por el usuario, cuyo valor depende del problema.

$$T_F = f_{best} * T_C \qquad (2)$$

La efectividad del método puede explicarse de la siguiente forma. Cuando el mejor individuo de la población tenga un fitness pobre (con valores grandes en valor absoluto), el rango de selección será mayor, debido a que el cambio en los valores de las constantes numéricas podrá ser 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 valores permitidos se reduzca progresivamente, favoreciendo una búsqueda local. Por este motivo es importante adaptar el valor del fitness de f_best para que se aproxime a cero cuando el algoritmo converja a la mejor solución. Usualmente el valor que se utiliza es el error entre la función deseada y la propuesta, así que la hipótesis resulta razonable.

Una vez aplicado el método, se conservará el nuevo conjunto de constantes solo si el cambio introducido mejora el rendimiento de la solución.

HILL CLIMBING (HC)

Hill Climbing es una técnica tradicional de optimización heurística que pertenece a la familia de los métodos búsqueda locales sin uso de derivadas. El algoritmo inicia con una solución arbitraria del problema, que en nuestro caso consiste en el conjunto de constantes involucradas en una solución de PG. Luego el algoritmo intenta mejorar dicha solución mediante perturbaciones aplicadas en varias iteraciones. Estas perturbaciones consisten en elegir una dirección al azar en el espacio de soluciones, y avanzando un paso tomado al azar entre 0 y el tamaño del paso máximo preestablecido (Eps_Max). Si la perturbación aplicada produce una mejor solución respecto a la original, ésta se conserva y será utilizada en la próxima iteración. El proceso se repite hasta que no se encuentran mejores soluciones en n iteraciones consecutivas.

La relativa simplicidad del algoritmo hace que sea popular su elección entre los algoritmos de optimización. Por este motivo, a pesar de ser un método de búsqueda local, se lo aplica para encontrar y evolucionar constates numéricas en la PG. Sin embargo su aplicación requiere evaluaciones adiciones de las soluciones, lo que aumenta considerablemente el costo computacional.

SIMULATED ANNEALING (SA)

Simulated Annealing (Kirkpatrick et al. 1983) es una evolución del método anterior. A diferencia de éste, SA es una heurística de búsqueda global ya que posee mecanismos que le permiten escapar de los óptimos locales. El mismo está inspirado en el recocido (annealing) en la metalurgia, una técnica que consiste en el calentamiento y posterior enfriamiento controlado de un material (en general aplicado en metales) para obtener estructuras cristalinas de baja energía.

En cada iteración del algoritmo se efectúa una perturbación en la solución similar a la descrita en HC. Luego, basados en una probabilidad que depende del valor de temperatura, se decide si la

solución resultante se conserva para la próxima iteración. La aceptación de soluciones malas con una cierta probabilidad es una propiedad fundamental de este tipo de heurísticas, ya que de esta manera se proporciona un mecanismo para escapar de la zona de óptimos locales.

El esquema de enfriamiento utilizado sigue un decremento exponencial en el tiempo, según la Ec. 3, donde T_0es la temperatura inicial y μ es el parámetro que permite controlar el descenso de temperatura.

$$T(t) = T_0 \ e^{\mu t} \qquad (3)$$

En el caso particular de PG, se adapta el concepto de temperatura y enfriamiento de SA, para encontrar mejores valores para las constantes numéricas. El inconveniente de la aplicación de este método para optimalizar las constantes es que, al igual que Hill Climbing, requiere evaluaciones adiciones que incrementan los tiempos de cálculo.

MULTI-DIMENSIONAL HILL CLIMBING (MDHC)

Este método recibe su nombre debido a que utiliza Hill Climbing para optimizar cada una de las constantes por separado (Fernandez, 1997). Multi-Dimensional Hill Climbing consiste en aplicar perturbaciones sobre el conjunto de constantes utilizadas por un individuo. Por cada constante se elige un número Δ elegido al azar dentro de un intervalo definido por el usuario, y se generan 2 perturbaciones de la siguiente manera: c-Δ y c+Δ. Por lo tanto, para un conjunto de k constantes se generan 2^k nuevas combinaciones que deben ser evaluadas para determinar cuál es la que produce un individuo con mejor fitness.

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. Para una población de tamaño N, la cantidad de evaluaciones adicionales requerida en cada generación es 3^k*N.

Debido a que la evaluación de los individuos suele ser la operación que consume más procesamiento en un algoritmo genético, normalmente se limita la cantidad de constantes que el método optimiza por vez, considerando como máximo 3 constantes, elegidas al azar. De este modo los tiempos de cálculo mejoran sin sacrificar la calidad de las soluciones.

DIFFERENTIAL EVOLUTION (DE)

Differential Evolution es un método que optimiza un problema iterativamente tratando de mejorar una solución candidata con respecto a un determinado fitness (Rainer and Kenneth, 1997). Una de las variantes más sencillas del algoritmo DE trabaja sólo en base a mutaciones de una población de soluciones candidatas. La mutación consiste en la construcción de vectores aleatorios, creados a partir de tres individuos elegidos al azar, llamados vectores objetivo (X_1,X_2,X_3), calculados según la Ec. 4. El parámetro F controla la tasa de mutación, y es básicamente un factor de escala.

$$X_i = X_1 + F * (X_2 + X_3) \qquad (4)$$

Esta variante de DE presenta una pobre estabilidad a repeticiones debido a que tiene una componente aleatoria muy fuerte. Sin embargo se la utiliza para optimizar las constantes numéricas en PG porque es un método sencillo de implementar y no tiene demasiado costo computacional.

EXPERIMENTACIÓN

En esta sección se presentan las pruebas realizadas con los algoritmos implementados. En todos los casos la metodología que se sigue consiste en resolver la regresión simbólica mediante PG, de donde se obtiene una solución inicial que luego se somete a un post-procesamiento. Este último paso consiste en una simplificación de las expresiones obtenidas utilizando propiedades algebraicas.

Se dan detalles para 3 casos, donde uno de ellos corresponde a un problema real. Todas las pruebas se corrieron bajo un entorno Linux, con un procesador Core i5 de 2.67 GHz y 3 GB de memoria RAM.

RASTRIGIN FUNCTION

La función de Rastrigin 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 de RS planteado radica en la obtención de constantes y no en la optimización global de un problema, se utilizó esta función definida para una sola variable independiente, y se cambiaron sus constantes como se muestra en la Ec. 5. De manera, la función así definida permite probar la eficacia y performance del sistema en casos difíciles. Los parámetros utilizados para las pruebas se muestran en la Tabla 1.

$$f(x) = 0.95x^2 - 5.5\cos(\pi x) \quad (5)$$

PARÁMETRO	VALOR	MÉTODO
GENERACIONES	500	-
TAMAÑO POBLACIÓN	500	-
CRUZAMIENTO	80%	-
MUTACIÓN	10%	-
ELITE	5%	-
SET DE FUNCIONES	+, -, *, /, sin, cos	-
PORCENTAJE DE POBLACIÓN AFECTADA A LA OPTIMIZACIÓN	80%	-
RANGO DE DEFINICIÓN DE CONSTANTES	[-5, 5]	-
ALTURA MÁXIMA DEL ÁRBOL	6	-
CONSTANTE DE TEMPERATURA (T_C)	0.002	MDHC
ITERACIONES CONSECUTIVAS MÁXIMAS (n)	15	HC, SA
TAMAÑO DEL PASO MÁXIMO (Eps_{max})	1	HC, SA
TEMPERATURA INICIAL (T_0)	2	SA
CONSTANTE DEL ESQUEMA DE ENFRIAMIENTO (μ)	0.05	SA
GENERACIONES MÁXIMAS	10	DE

Tabla 1 – Parámetros utilizados para la obtención de la función Rastrigin.

NUMERIC MUTATION

Utilizando los parámetros de entrada descritos anteriormente, la mejor solución obtenida con este método se muestra en la Fig. 3. Luego de un post-procesamiento, la ecuación final se muestra en la Ec. 6. El fitness que obtuvo el mejor individuo de esta prueba fue igual a 0.0649662 en un tiempo total de 146 segundos. En la Fig. 4 se puede ver la curva correspondiente al mejor individuo y la curva de la función original.

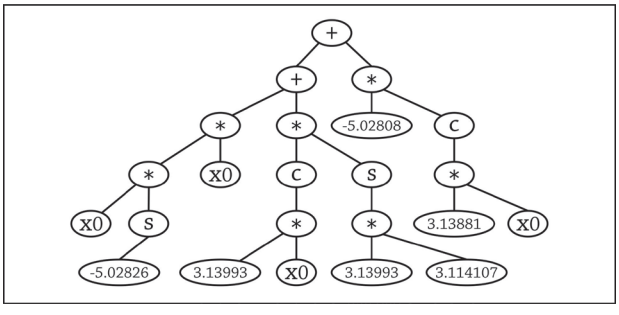

Fig. 3 - Mejor individuo obtenido para la función Rastrigin, utilizando PG con NM

$$f(x) = \cos(3.13881x) * 3.13881x^2 - 0.422558\cos(3.13993x) \quad (6)$$

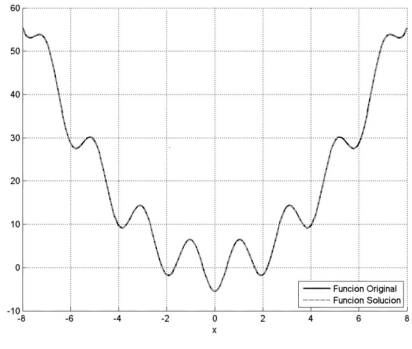

Fig. 4 - Función de salida vs función original para NM.

HILL CLIMBING

Con este método se obtuvo un individuo final cuya estructura de árbol se muestra en la Fig. 5, y la Ec. 7 corresponde a su expresión simplificada.

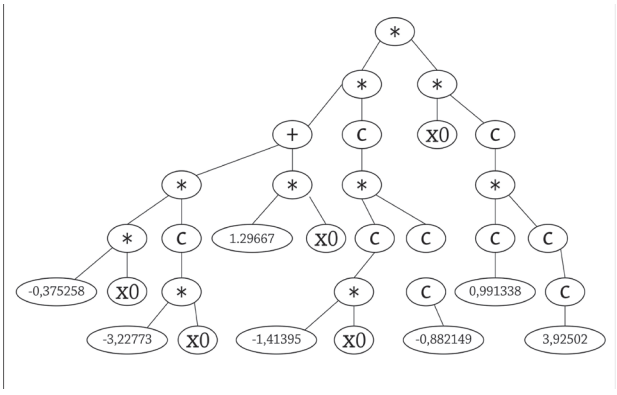

Fig. 5 - Mejor individuo obtenido con HC.

$$f(x) = -0.3432x^2(\cos(3.13881x) - 3.4554x)\cos(0.8047\cos(1{,}4139x)) \quad (7)$$

La ejecución se llevó a cabo en 8337 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 Fig. 6. Si bien se observa que el algoritmo sigue la tendencia general de la función, los resultados no son tan buenos como los obtenidos con el método anterior.

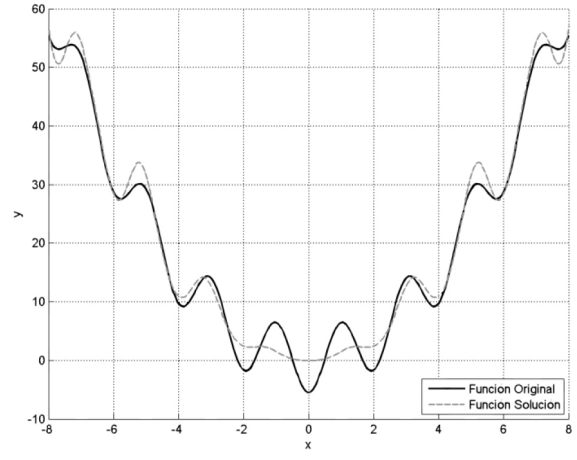

Fig. 6 - Función de salida vs función original para HC.

SIMULATED ANNEALING

La representación del mejor individuo obtenido con este método y su ecuación simplificada se muestran en la Fig. 7, y la Ec. 8 respectivamente.

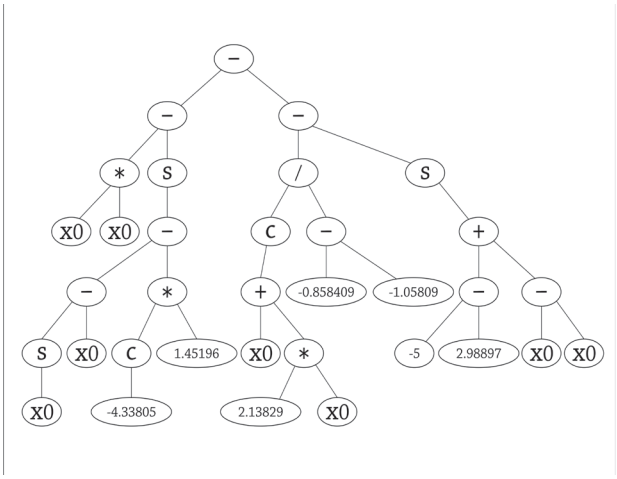

Fig. 7 - Mejor individuo obtenido con SA.

$$f(x) = x^2 - sen(-x + sen(x) + 0.5309) - 5.0079\cos(3.1382x) - 0.9909 \quad (8)$$

Diferente al caso anterior, este método pudo aproximar bien los datos de entrada en su mayoría, logrando un fitness de 1.09076 en 11150 segundos. La curva resultante y los datos experimentales se pueden ver en la Fig. 8.

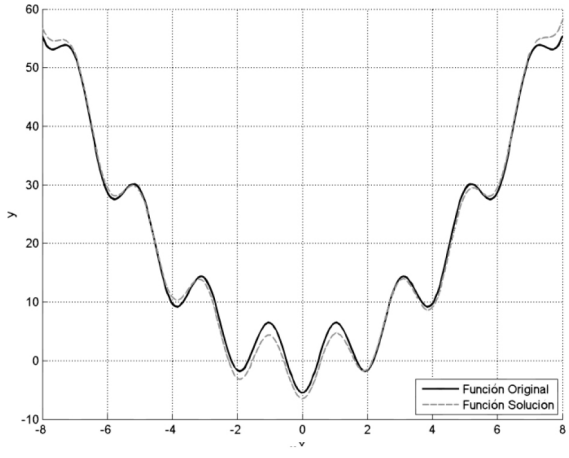

Fig. 8 - Función de salida vs función original para SA.

DIFFERENTIAL EVOLUTION

Con este método se obtuvo el individuo representado en la Fig. 9, cuya ecuación simplificada se muestra en la Ec. 9.

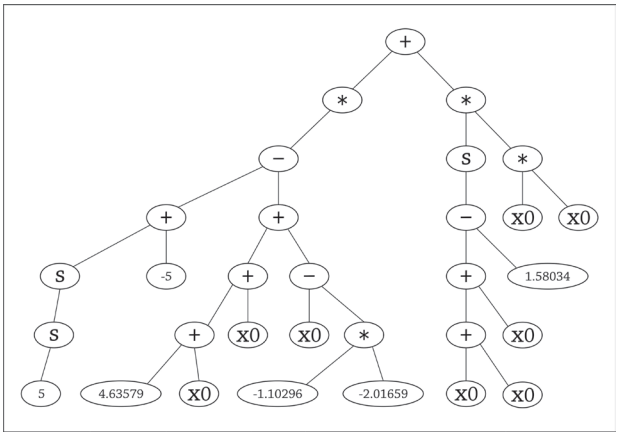

Fig. 9 - Mejor individuo obtenido con DE.

$$f(x) = -sen(1.5803 - 3x) + x^2 - sen(3x + 8.2301) \quad (9)$$

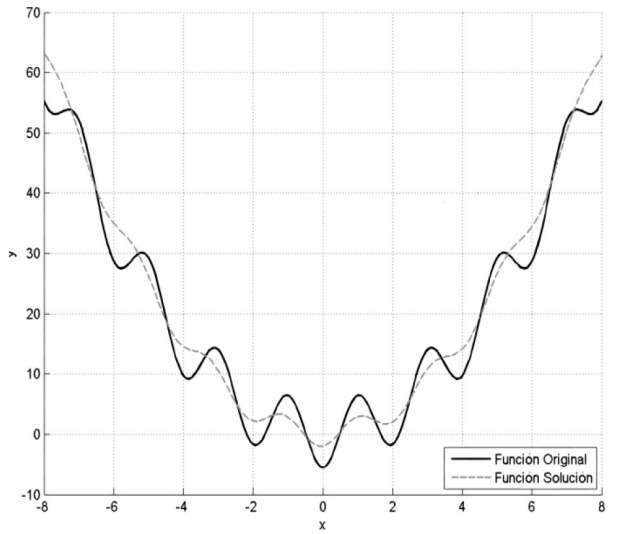

Fig. 10 - Función de salida vs función original para DE.

A partir de la Fig. 10 y el valor de fitness obtenido se concluye que el método no pudo obtener una solución aceptable comparada con NM o SA.

MULTI DIMENSIONAL HILL CLIMBING

Con este método se obtuvo el individuo de la Fig. 11 y como salida la Ec. 10 simplificada.

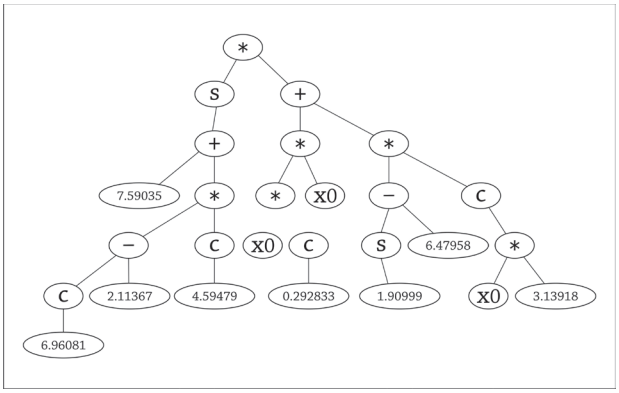

Fig. 11 - Mejor individuo obtenido con MDHC.

$$f(x) = 0.9519x^2 - 5.5048\cos(3.1391x) \quad (10)$$

El individuo consiguió un valor de fitness de 0.0778178 en 1506 segundos como se puede observar en la Fig. 12.

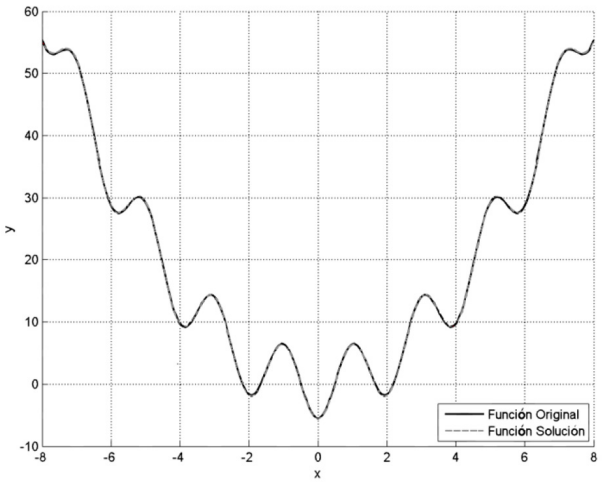

Fig. 12 - Función de salida vs función original para MDHC.

MCCORMICK FUNCTION

McCormick es otra función utilizada como benchmark para problemas de optimización (Hyndman,

2005). La misma está definida por la Ec. 11, cuya gráfica se muestra en la Fig. 13.

$$f(x, y) = -\operatorname{sen}(x+y) + (x-y)^2 - 1.5x + y + 2.5y + 1 \quad (11)$$

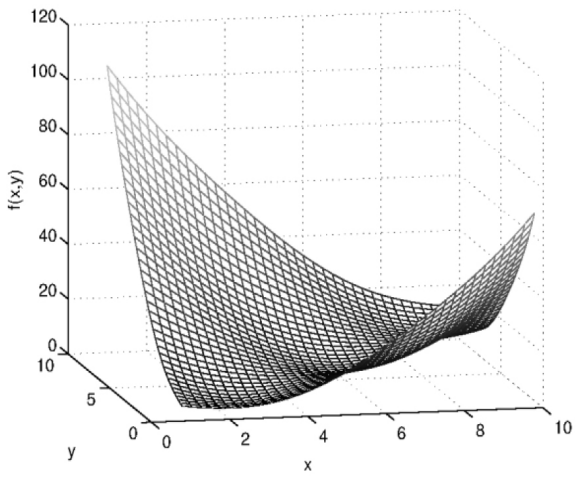

Fig. 13 - Gráfica de McCormick Function.

PARÁMETRO	VALOR
GENERACIONES	200
TAMAÑO POBLACIÓN	150
CRUZAMIENTO	80%
MUTACIÓN	30%
ELITE	5%
SET DE FUNCIONES	+, -, *, /, sin, cos
CONSTANTE DE TEMPERATURA (Tc)	1.6

Tabla 2 - Parámetros utilizados para la función McCormick.

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 utilizados se muestran en la Tabla 2.

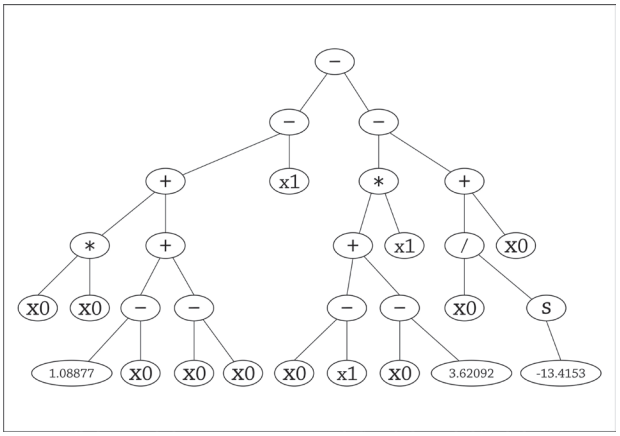

Fig. 14 - Mejor individuo obtenido.

$$f(x) = -x - 2xy - 1.2212x + y^2 + 2.6209y + 1.0887 \quad (12)$$

El mejor individuo obtenido corresponde al árbol de la Fig. 14, y su expresión simplificada se muestra en la Ec. 12.

Este individuo consiguió un fitness igual a 1.40393 en 7980 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 mismo motivo no fueron encontradas las componentes más pequeñas o menos significativas (término que incluye la función seno). Para solucionar este problema se hizo una segunda corrida del algoritmo, pero esta vez utilizando el

$$f(x) = -0.3999\operatorname{sen}(x+y) - 0.1999x - 0.1999y - 0.1998 \quad (13)$$

residuo, el cual se calcula como la diferencia entre los valores calculados con la solución obtenida y los datos experimentales. De este modo el algoritmo descubre aquellas componentes de la solución no

$$f(x) = -0.3999\operatorname{sen}(x+y) - 0.1999x - 0.1999y - 0.1998 \quad (14)$$

encontradas en la primera etapa (Ec. 13).

Para construir la solución definitiva, se deben combinar los resultados obtenidos en ambas etapas, sumando ambas expresiones. En la Ec. 14 se muestra la solución final.

El error en las constantes encontradas es menor a 0.15, excepto por la del coeficiente del la función seno, que es ligeramente mayor (0.66). Esta diferencia se debe a que la función seno tiene un valor máximo en 1, el cual es 2 órdenes de magnitud menor que el resto de los términos. Para mejorar este resultado se podrían utilizar métodos de ajuste local, como regresión lineal, o métodos con derivada, si hay indicios de que requieren constantes dentro de la función seno (en lugar de sólo x+y). Es importante destacar que incluso sólo con la primera fase (Ec. 12), se obtiene una función muy próxima a la original.

NOMBRE	EXPRESIONES	RMSE
3-BINOMIAL	$f(x) = x^3 + 3x^2 + 3x + 1$ $f_{PG}(x) = x^3 + 3.00826x\,(x+1) + 0.949622$	0.0831
IDENTIDAD TRIGONOM	$f(x) = \cos(2x)$ $f_{PG}(x) = \sin(2x - 4.59955)$	0.0296
POLINOMIO DE GRADO 2	$f(x) = 2.7128x^2 + 3.141636x$ $f_{PG}(x) = 2.71967x^2 + 3.141691x$	0.0541
POLINOMIO DE GRADO 3	$f(x) = x^3 - 0.3x^2 - 0.4x - 0.6$ $f_{PG}(x) = 0.99856x^3 + 0.29709x^2 -$ 03313946x - 0.687809	0.1504
BENCHMARK 1	$f(x) = x \sin(\pi x)$	0.0009
	$f_{PG}(x) = \sin(3.1416x)x$	
BENCHMARK 2	$f(x) = 2.4 \sin(x^2 + 0\text{-}76x + \pi)$ $f_{PG}(x) = -2.52449 \sin(x^2 + 0.757469x)$	0.0874
RASTRIGIN MODIFICADA	$f(x) = 0.85x^2 + 11.5 \cos - (\pi x)$ $f_{PG}(x) = -0848222x^2 - 11.4433 \cos(3.12437x)$	0.0976
SENO AMORTIGUADO	$f(x) = \frac{3}{2}\, e \sin(\pi x+\pi)$ $f_{PG}(x) = -e^{-0.385643x} (\sin(3.12437x) + 0.252533)$	0.0835
BOOTH FUNCTION	$f(x,y) = (x+2y-7)^2 + (2x+y-5)^2$ $f_{PG}(x,y) = 5.00117(x^2+y^2) + 8.00234xy$ $+ 34.2126x + 38.2126y + 74$	1.7835

Tabla 3 – Resumen de resultados obtenidos en otras pruebas de nchmark.

OTRAS FUNCIONES DE BENCHMARK

Además de las funciones de prueba detalladas anteriormente, se hicieron pruebas sobre otros problemas tomados de otros trabajos (Byrne et al., 2009; Zhao et al., 2012). En la Tabla 3 se listan los benchmarks probados, las fórmulas resultantes después de aplicar el post-procesamiento a la mejor solución obtenida, y el nivel de error logrado.

Pasajeros de Aerolínea Internacional

Este caso corresponde a una prueba con datos reales publicado para su reproducción y análisis (Hyndman et al., 2005). El problema consiste en una serie de tiempo, en donde se involucra la cantidad de

PARÁMETRO	VALOR
GENERACIONES	450
TAMAÑO POBLACIÓN	180
CRUZAMIENTO	55%
MUTACIÓN	10%
ELITE	5%
SET DE FUNCIONES	+, -, *, /, sin, cos. pow, sqrt, log
CONSTANTE DE TEMPERATURA (Tc)	1

Tabla 4 - Parámetros utilizados en el problema de los Pasajeros de Aerolínea Internacional.

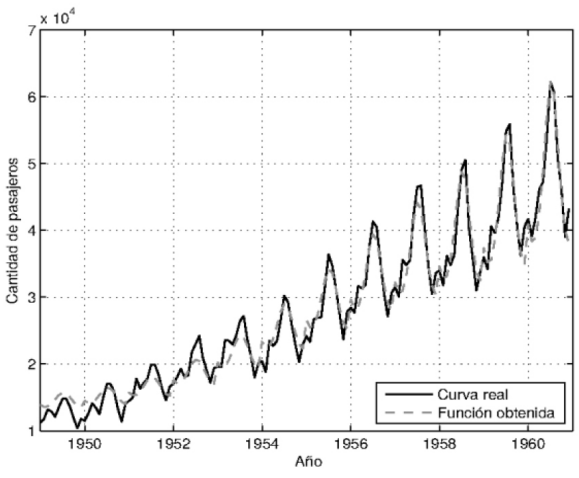

Fig. 15 - Función de salida vs función original.

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 fecha se representa mediante dos variables, mes y año.

Al igual que en el caso anterior, la prueba se realizó con el método Multi-Dimensional Hill Climbing, utilizando los parámetros listados en la Tabla 4.

En la Fig. 15 se pueden observar una comparativa entre la serie de tiempo original (línea llena) y la estimada con la mejor solución obtenida con programación genética (línea punteada). Como se puede observar, la solución encontrada permite estimar correctamente la cantidad de pasajeros. Así mismo, en la Fig. 16 se puede observar la representación de árbol de la expresión resultante. Al recorrer el árbol en preorden se puede obtener

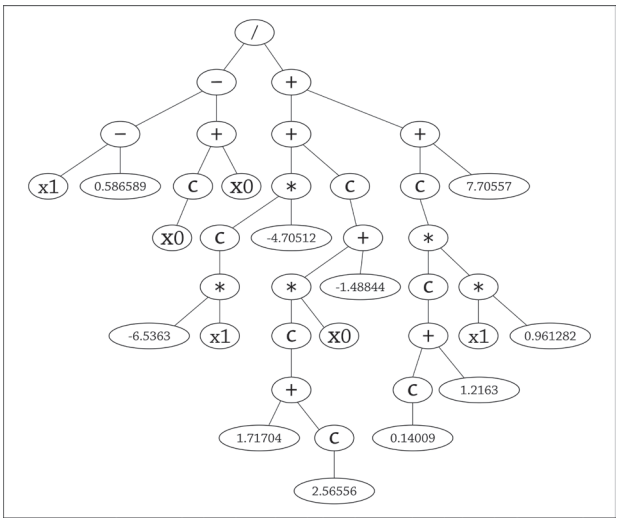

Fig. 16 - Árbol del mejor individuo obtenido

la función que define a esta serie de tiempo.

Esta prueba obtuvo un fitness igual a 19.9834 en 250 segundos, resultados en los cuales se ve reflejada la complejidad del problema. La serie de tiempo presenta componentes autorregresivos (Hyndman et al., 2005), es decir que incorporar datos correspondientes a instantes de tiempo anteriores como variables de entrada puede mejorar los resultados. Sin embargo en nuestro caso no fue necesario incluirlo debido a que el resultado obtenido es suficiente para demostrar el potencial que tiene esta herramienta.

CONCLUSIONES

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, aunque las componentes pequeñas (residuo) no siempre fueron encontradas.

Las soluciones encontradas utilizando los métodos NM, MDHC y SA son las que arrojaron menor error, y permitieron aproximar los datos de entrada en forma efectiva. Sin embargo se detectó una diferencia significativa con los tiempos de ejecución y los requerimientos de procesador y memoria. Al requerir demasiadas evaluaciones a la función fitness, los métodos generacionales (SA, HC y MDHC) requieren más tiempo para lograr obtener una buena solución, por lo que resultan poco eficientes en ciertas aplicaciones donde el tiempo de resolución es un factor importante.

Cabe destacar que entre los algoritmos implementados, NM es el de menor complejidad, que demostró ser muy eficiente en términos de tiempo de ejecución, arrojando buenos resultados en problemas simples, pero no asegura buenos resultados en casos complejos. DE también resultó ser eficiente en procesamiento, pero tampoco asegura resultados, siendo sobrepasado en eficiencia y resultados por NM.

En cuanto a la estabilidad del programa, se encontraron respuestas correctas en 1 de cada 5 veces que se ejecuto el algoritmo, y en varios casos fue necesario repetir el proceso en varias

etapas, haciendo un análisis por residuos para encontrar un resultado correcto. Sin embargo, 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.

En cuanto al Framework utilizado, demostró tener la robustez necesaria para la aplicación de problemas de Regresión Simbólica de distintas características, permitiendo obtener soluciones de buena calidad con tiempos de cálculo 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.

REFERENCIAS

John R. Koza. "Genetic Programming on the Programming of Computers by Means of Natural Selection". MIT Press, Cambridge, MA, USA, (1998).

Riccardo Poli, William B. Langdon, Nicholas F. McPhee and John R. Koza, "A Field Guide to Genetic Programming", Edición Ilustrada, Lulu .com, (2008).

Diego Santochi and Josefina Lenis, "Plataforma de Programación Genética en C++". Tesis de grado de la carrera de Ingeniería en Computación de la Universidad Nacional de Tucumán, (2011).

Federico Navarro, Luis Remis, Diego Santochi, Josefina Lenis, Adrian Will and Sebastián Rodríguez, "Programación Genética y Aplicaciones a Robótica: Cortadora de Pasto y Resolución de Laberintos", en Actas de las VII Jornadas Argentinas de Robótica, Olavarría, (2012).

David E. Goldberg and Jon Richardson, "Genetic algorithms with sharing for multi-modal function optimization", en Genetic Algorithms and Their Applications: Proceedings of the Second International Conference on Genetic Algorithms, 44 – 49, ICGA, (1987).

Yossi Borenstein and Riccardo Poli, "Empirical approach for theory of randomized search heuristics", Proceeding of the Second International Conference on Genetic Algorithms, (2006).

X. Yu and M. Gen, "Introduction to Evolutionary Algorithms", Springer, (2010).

Matthew P. Evett and Thomas Fernandez, "Numeric Mutation: Improved Search in Genetic Programming", in FLAIRS Conference, 106–109, (1998).

S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," SCIENCE, vol. 220, no. 4598, 671–680, (1983).

Thomas Fernandez, "The evolution of numeric constants in genetic programming", (1997).

Rainer Storn and Kenneth Price, "Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces" Journal of global optimization, vol. 11, no. 4, 341–359, (1997).

Rob J. Hyndman, "Time series data library", (2005). URL: http://data.is/TSDLdemo, accedido el 24/08/2014.

K. Tang, X. Yao, P. N. Suganthan, C. MacNish, Y. P. Chen, C. M. Chen, and Z. Yang, "Benchmark Functions for the CEC'2008 Special Session and Competition on Large Scale Global Optimization", Technical Report, Nature Inspired Computation and Applications Laboratory, USTC, China, (2007).

Jonathan Byrne, Erik O Neil, and Anthony Brabazon. "Analysis of constant creation techniques on the binomial-3 problem with grammatical evolution", IEEE Congress on Evolutionary Computation, 568-573, (2009).

Li Zhao, Lei Wang, and Du wu Cui. "Hoeffding bound based evolutionary algorithm for symbolic regression". Engineering Applications of Artificial Intelligence, vol. 25, no 5, 945–957, (2012).

Fecha de Presentación: 15/05/15
Fecha de Aprobación: 08/06/15

[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.