MÉTODO BASADO EN PROGRAMACIÓN GENÉTICA PARA LA SOLUCIÓN SIMBÓLICA DE ECUACIONES DIFERENCIALES

A METHOD BASED ON GENETIC PROGRAMMING TO FIND SYMBOLIC SOLUTIONS OF DIFFERENTIAL EQUATIONS

Angélica María Ramírez Botero[1], Julio Hernando Vargas[2], César Augusto Acosta Minoli[3]

1. Universidad del Quindío, Grupo GEDES, Email: amramirezb@uniquindio.edu.co.
2. Universidad Tecnológica de Pereira, Email: jhvargas@utp.edu.co.
3. Universidad del Quindío, Grupo GEDES, Email: cminoli@uniquindio.edu.co

Recibido: 30 de Julio de 2014
Aceptado: 01 de Noviembre de 2014
*Correspondencia del autor:Grupo GEDES Universidad del Quindío Armenia Colombia. Email: amramirezb@uniquindio.edu.co.

RESUMEN:

En este trabajo se realizó un estudio sobre el uso de un método basado en Programación Genética para resolver ecuaciones diferenciales. En este estudio se describen las principales características del método exponiendo los fundamentos que soportan el desarrollo del algoritmo, desde su explicación conceptual hasta su implementación en el lenguaje de programación Python. Posteriormente, se presentan tres problemas seleccionados de tal forma que permiten ilustrar diferentes características del uso del método, considerando un problema de valor inicial, un problema de valor de frontera no-lineal y un problema numéricamente inestable. Finalmente, el trabajo concluye presentando una discusión sobre las ventajas y desventajas del método de programación genética a la luz de los problemas anteriormente expuestos y presentando una serie de preguntas las cuales motivan el desarrollo de un posterior trabajo de investigación.

Palabras claves: Programación genética, regresión simbólica, métodos numéricos, ecuaciones diferenciales.

ABSTRACT

In this study, a method based on Genetic Programming (GP) to solve differential equations was carried out. We first described the fundamentals of the GP method from its conceptual insights until its implementation in Python. Then we used the method to solve three problems in ODEs to illustrate some advantages and disadvantages from a computational and a mathematical point of view. The problems solved included an initial value problem, a non-linear boundary value problem and a numerically unstable problem. Results show some advantages concerning the use of the method for non-linear and ill-conditioned problems and allow us to propose several questions for future research.

Keywords: Genetic programming, symbolic regression, numerical methods, differential equations.

1. INTRODUCCIÓN

En el caso de los métodos computacionales para resolver ecuaciones diferenciales, recientemente se viene desarrollando una serie de métodos basados en lo que se denomina computación evolutiva, los cuales permiten solucionar la ecuación diferencial en forma simbólica, es decir, mediante la búsqueda de una fórmula analítica cerrada que aproxime o que encuentre de forma exacta la solución del problema.

La computación evolutiva corresponde a un método de optimización estocástico cuyas reglas de búsqueda se fundamentan en los principios de selección natural. Este enfoque considerado en la literatura como biológicamente inspirado fue propuesto principalmente por Holland, ver (Holland,1975). La computación evolutiva consiste entonces, de un proceso basado en la evolución de un número de posibles soluciones del problema las cuales se modifican o evolucionan a través de ciertos operadores denominados genéticos tales como la réplica, el cruzamiento y la mutación, ver (Coello, 2006).

Los algoritmos evolutivos comparten las siguientes características, ver (Back, 2000): (i) Su éxito radica en el patrón emergente que surge de la evolución de la población de individuos. Usualmente cada individuo, representa o codifica un punto en el espacio de potenciales soluciones del problema a resolver. (ii) Los descendientes de los individuos son generados por procesos aleatorios que intentan modelar la mutación y la recombinación. (iii) Con el fin de evaluar a los individuos en su ambiente, una medida de capacidad de adaptación a su entorno se debe asignar a cada individuo. De acuerdo con esta medida, el proceso de selección favorecerá a los mejores individuos a reproducirse con más frecuencia que aquellos con un peor desempeño.

Dentro de los algoritmos de computación evolutiva se destaca la programación genética, propuesta originalmente por Koza (Koza, 1992). Está técnica se diferencia de un algoritmo genético puro en dos aspectos: El primero es que la programación genética codifica las soluciones en forma de árbol y no como cadena de números de enteros y el segundo es que la programación genética permite que las mejores soluciones de una generación pasen de forma directa a la generación siguiente, esto es lo que se considera en la literatura como elitismo, Ver (Ashlock, 2006). Esta técnica ha sido utilizada de forma exitosa en problemas tales como: generación automática de código de programación, regresión simbólica, descubrimiento de identidades trigonométricas, control de robots y hormigas artificiales entre otros.

Con relación a la teoría que permite comprender la convergencia de los algoritmos evolutivos, Schmitt y colaboradores, (Schmitt, 2006) demuestran convergencia asintótica a un óptimo global para una familia de sistemas de programación genética donde la población estaba conformada por un número fijo de individuos cada uno de tamaño arbitrario. El modelo teórico que se usa para estudiar la convergencia para el caso de la programación genética donde los individuos son árboles de tamaño arbitrario, son las cadenas de Markov no homogéneas en el espacio de vectores de dimensión infinita.

Por otra parte, con respecto a la forma en que la programación genética realiza la búsqueda en el espacio de soluciones Daida y Hilss (Daida, 2003) logran identificar cómo la estructura de datos denominada árbol la cual se usa para codificar los individuos en programación genética, tiene una influencia en la manera como el algoritmo realiza la búsqueda en el espacio de soluciones. A este fenómeno intrínseco de la programación genética se le denomina en la literatura como dificultad estructural (Foster, 2005), la cual impide una búsqueda efectiva, imparcial, no sesgada en el espacio de búsqueda, particularmente en la búsqueda de soluciones cuya estructura de datos esté determinada por árboles angostos o por árboles muy anchos. La búsqueda de soluciones prácticas a este problema, es en la actualidad un tema abierto de investigación, ver (Thorhauer, 2013), (O'Neill, 2003, 2003b).

En el campo de la aplicación de la computación evolutiva a las ecuaciones diferenciales, se encuentran varios trabajos desarrollados, ver Mastoraskis y colaboradores (Mastorakis,2006), Asif-Zahoor et al. (Azif, 2009) y Jebari et. al. (Jebari, 2013) entre otros. Se destaca el trabajo de Tsoulos y Lagaris (Tsoulos et. al., 2006) los cuales desarrollaron un método basado en programación genética para la solución simbólica de ecuaciones diferenciales ordinarias y parciales.

Este método crea generaciones de expresiones simbólicas que representan posibles soluciones de la ecuación diferencial en forma analítica cerrada, dichas ex-

presiones se construyen mediante una gramática libre de contexto en la forma Backus-Naur. Los ejemplos ilustrativos de su trabajo demuestran la potencialidad del método para la solución de ecuaciones diferenciales. Sin embargo su trabajo no abarca problemas que permitan abrir una discusión sobre las posibles ventajas y desventajas que presenta esta metodología.

Por lo tanto, el presente artículo tiene por objetivo realizar un estudio sobre el uso de un método basado en Programación Genética para resolver ecuaciones diferenciales partiendo de los estudios de Tsoulos y Lagaris (Tsoulos,2006).

Este documento se compone de tres secciones. En la siguiente sección, se describe el método de programación genética para la solución de ecuaciones diferenciales ordinarias. Posteriormente, se presentan tres problemas seleccionados de tal forma que permiten ilustrar diferentes características del uso del método, considerando un problema de valor inicial, un problema de valor de frontera no-lineal y un problema numéricamente inestable. Finalmente, se concluye presentando una discusión sobre las ventajas y desventajas del método de programación genética a la luz de los problemas anteriormente expuestos.

2.METODO DE PROGRAMACIÓN GENÉTICA PARA ODEs

Para resolver una ecuación diferencial mediante regresión simbólica por medio de Programación Genética (GP) primero se debe codificar una posible solución en un árbol de expresiones el cual denotará un individuo. Luego se crea una población de estos individuos a la cual se le aplican los operadores genéticos de forma iterativa hasta obtener convergencia. Ver (Tsoulos, 2006) y Koza (Koza,1992).

2.1 Codificación de un individuo

Para codificar un individuo, el algoritmo requiere una gramática libre de contexto en la Forma Bakus Naur (BNF) (ver (De Castro, 2004) y (Tsoulos, 2006). Un cromosoma en programación genética es un árbol que denota una expresión bien formada de la gramática BNF. Por lo tanto, en el caso de ecuaciones diferenciales se considera una gramática que genere expresiones analíticas generadas de la siguiente gramática:

```
S       ::=          <expr>                    (0)
<expr>  ::=  <expr> <op> <expr>                (0)
                   | ( <expr> )                (1)
                   | <func> ( <expr> )         (2)
                   | <digit>                   (3)
                   | x0                        (4)
                   | x1                        (5)

<op>    ::=          +                         (0)
                   | -                         (1)
                   | *                         (2)
                   | /                         (3)

<func>  ::=          sin                       (0)
                   | cos                       (1)
                   | exp                       (2)
                   | log                       (3)

<digit> ::=          0                         (0)
                   | 1                         (1)
                   | 2                         (2)
                   :                           :
                   :                           :
                   :                           :
                   | 9                         (9)
```

El símbolo *S* denota el inicio del proceso, *expr* significa una expresión, la cual puede ser cualquiera de las seis formas posibles definidas, *op* denota las operaciones binarias disponibles, *func* las funciones de un sólo argumento y *digit* los dígitos del 0 al 9.

La construcción de cualquier individuo de la población inicial es un proceso aleatorio. Comienza con la elección de un nodo inicial o raíz, y luego, como para cada categoría de la gramática existe una lista, entonces se hace un sorteo para elegir una de ellas. Nuevamente se elige al azar un elemento de esa lista, el cual puede ser, dependiendo de la elección, un vértice interno o un vértice terminal (hoja). Si el elemento elegido es una operación binaria, entonces se formará un subárbol binario que siempre pondrá en la primera elección el hijo izquierdo y en la segunda el hijo derecho. Si el elemento elegido al azar es una función, el vértice sólo despliega un hijo el cual puede volverse un operador binario o unario dependiendo del sorteo. Si en la elección, el resultado es un dígito o una variable, este nodo se convertirá en una hoja. La figura muestra un individuo producido mediante el proceso anterior.

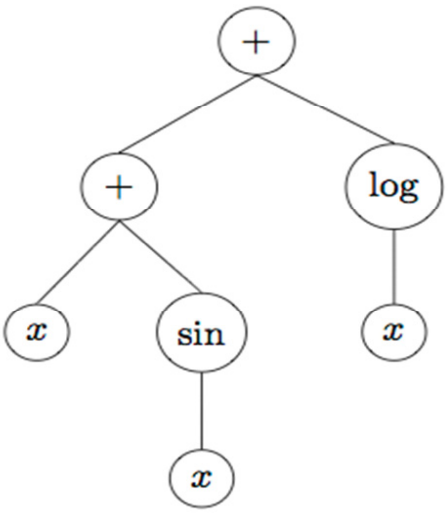

Figura 2.1 Ejemplo de un individuo codificado generado con la gramática.

El árbol de la figura anterior produce la siguiente fórmula en notación infija:

$$((x + \sin(x)) + (\log(x)))$$

2.2 Descripción del algoritmo

2.2.1 Inicialización

Se inicia con la generación aleatoria de individuos (árboles) hasta obtener una población de tamaño *g*. En esta fase también se inicializan los valores para la tasa de réplica y la tasa de mutación. La tasa de réplica denota la fracción del número de árboles que irán sin ningún cambio a la próxima generación. Esto es

$$Probabilidad\ de\ Cruce = 1 - t_r,$$

donde t_r es la tasa de réplica.

2.2.2 Función de adaptación

Primero la ecuación diferencial ordinaria se debe expresar como:

$$f(x,y,y^{(1)},\ldots,y^{(n-1)},y^{(n)}) = 0,\ x \in [a,b]$$

donde $y^{(i)}$ denota la derivada de orden *i* de *y*. Por su parte, las condiciones iniciales o de frontera se deben escribir en la forma:

$$g_i\,(x,y,y^{(1)},\ldots,y^{(n-1)}) \mid_{x=ti} = 0,\ i = 1,\ldots,n$$

donde t_i es uno de los extremos del intervalo del dominio, *a* ó *b*. Luego, para calcular la función de adaptación de la población se deben realizar los siguientes pasos:

1. Elegir *N* puntos, $(x_0,x_1,\ldots,x_{n-1})$ en el dominio $[a,b]$.
2. Para cada árbol *i* :

(a) Construir y evaluar la expresión en notación infija $M_i\,(x)$ en los puntos x_j.

(b) Calcular y evaluar en los puntos x_j, las n derivadas de $M_i\,(x)$.

(c) Calcular la identidad:

$$E(M_i) = \sum_{j=0}^{N-1} w_j\,(f(x_j,M_i^0(x_j),\ldots,M_i^{(n)}(x_j)))^2$$

(d) Calcular una función de penalidad asociada $P(M_i)$ la cual depende de las condiciones iniciales o de frontera y tiene la forma:

$$P(M_i) = \lambda \sum_{k=1}^{n} g_k^2\ (x,M_i,M_i^{(1)},\ldots,M_i^{(n-1)}) \mid_{x=tk})$$

donde λ es un número positivo.

(e) Calcular el valor de adaptación del árbol como:

$$v_i = E(M_i) + P(M_i).$$

De esta forma se evalúan todos los árboles de la población y se ordenan de acuerdo a su valor de adaptación.

2.2.3 Evaluación de las derivadas

La evaluación de las derivadas de forma simbólica se hace mediante la librería SymPy de Python, para más información consultar *http://sympy.org/en/index.html*

2.2.4 Operadores Genéticos

Los operadores genéticos utilizados son: la réplica, el cruce y mutación. Una vez ordenados los individuos de la población en función de su valor de adaptación, la réplica consiste en elegir una cantidad de mejores individuos de la población ordenada en cada generación para trasmitirla sin modificaciones a la siguiente generación. El cruce se aplica para crear nuevos árboles en cada generación, los cuales reemplazarán a los últimos individuos de la lista ordenada. En esta operación, cada pareja de padres es seleccionada, como en el ejemplo de la Fig. 2.2, luego se elige de cada padre un nodo al azar para el corte, ver Fig. 2.3 y posteriormente se hace el intercambio del subárbol o material genético como se muestra en la figura 2.4, (García et al., 2009).

Los padres son seleccionados mediante selección de torneo: Primero, se crea un grupo de individuos $K \geq 2$, seleccionados aleatoriamente de la población actual. Posteriormente, los individuos con las mejores funciones de adaptación en el grupo se seleccionan, los demás son descartados. El último operador genético usado es la mutación, el cual cambia aleatoriamente un nodo del árbol (enumerados previamente) de manera aleatoria.

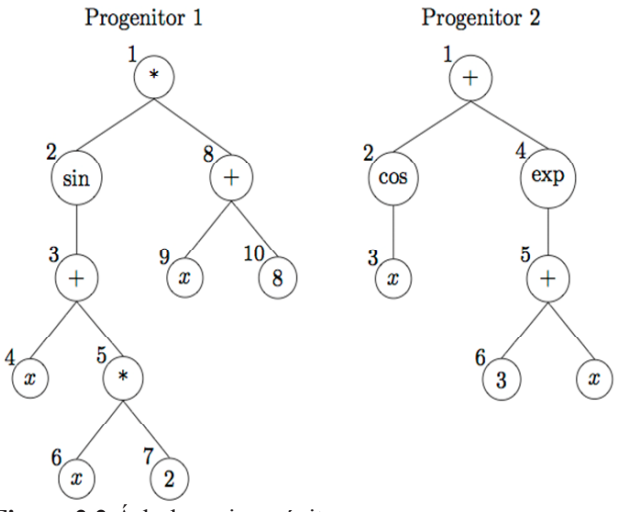

Figura 2.2 Árboles primogénitos.

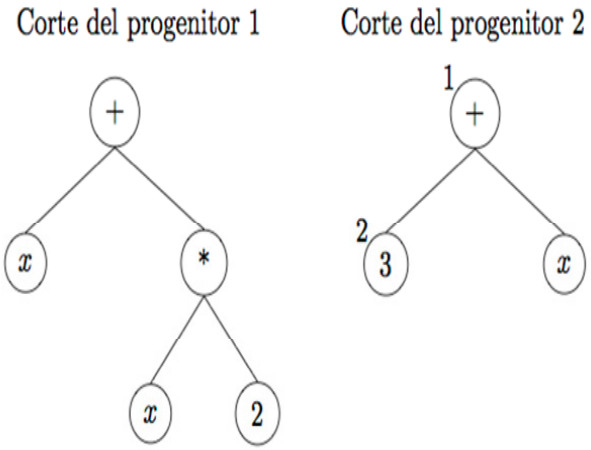

Figura 2.3 Subárboles producto de la selección al azar de los nodos 3 y 5 de los progenitores 1 y 2 respectivamente (Cruce en un punto)

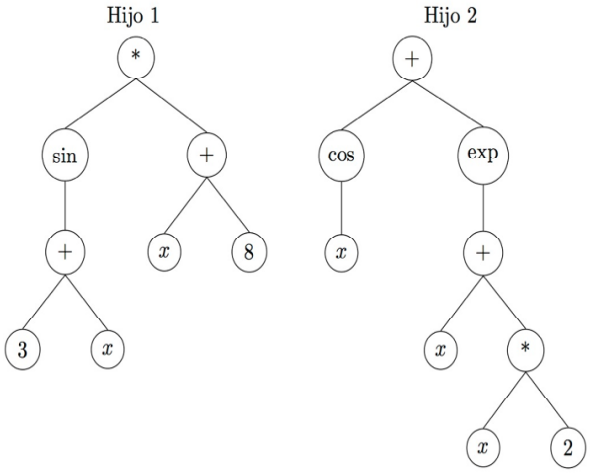

Figura 2.4 Árboles hijo producto del cruce de dos individuos en un punto.

Con respecto a cómo se aplican los operadores genéticos, en cada generación se realizan los siguientes pasos:

1. Los árboles son ordenados con respecto a sus funciones de adaptación, de tal manera que el mejor individuo queda al principio de la lista y el peor individuo, al final de ella.
2. Del *100*% de la población inicial, el 10% de los mejores individuos son seleccionados automáticamente para la siguiente generación (*réplica*). El 90% restante es seleccionado de la siguiente forma:

i) De la lista originalmente ordenada de los individuos, se toma un porcentaje de los mejores individuos, en nuestro caso del *40*%, con los cuales se hace *por torneo* (se toman dos individuos aleatoriamente y se elige el mejor) para formar el primer 45% de una lista a la que llamaremos *población intermedia*.

ii) De esta población intermedia, se eligen aleatoriamente dos padres, para realizar el operador de *cruce*, y así formar el otro 45% restante.

Los nuevos individuos reemplazarán a los peores de la población al final del operador cruce.

3. El operador de *mutación* es aplicado a un porcentaje de la nueva población, exceptuando aquellos individuos que fueron seleccionados en la operación réplica. La importancia de este operador es que genera nuevos individuos y a su vez, nuevos espacios de búsqueda en aquellos casos en que la población tiende a ser homogénea.

2.2.5 Finalización del algoritmo

Los operadores genéticos son aplicados a la población creando nuevas generaciones, hasta alcanzar un número máximo de generaciones o hasta obtener el mejor árbol de la población según una tolerancia preestablecida.

3. EXPERIMENTOS NUMÉRICOS (RESULTADOS)

Este sección tiene por objetivo presentar el desempeño del algoritmo en la solución de ecuaciones diferenciales, por lo tanto, se presentan tres problemas seleccionados que ilustran el uso del método.

Con respecto a los parámetros del algoritmo y de los operadores genéticos utilizados en cada uno de los problemas, estos fueron seleccionados con base a los experimentos numéricos reportados en el libro de Koza (Koza, 1992) y en el artículo de Tsoulos y Lagaris (Tsoulos, 2006). De esta forma, se utilizó una tasa de réplica del 10% y por tanto un 90% de

cruzamiento, como también un 10% de tasa de mutación. Diversas investigaciones demuestran como la modificación de estos parámetros pueden influir en el desempeño y convergencia del algoritmo, ver (Koza, 1992), (Tsoulos, 2006), (Fagan, 2012). Sin embargo tales modificaciones y su influencia no son objetivo de este estudio.

El tamaño de la población se configuró en 500 y los individuos fueron codificados con árboles de un máximo de 10 niveles de profundidad. El tamaño de la población es un parámetro crítico y fue tomado de (Koza, 1992). Según Tsoulos y Lagaris (Tsoulos, 2006), una población con pocos individuos debilitan la capacidad del método para la construcción de la mejor solución, un número grande de individuos genera excesivos sobre costos computacionales.

Los experimentos se realizaron en un computador con procesador intel 3.4GHz quad-core y 12GB de memoria RAM, sistema operativo OS X versión 10.9.5. El parámetro de penalización fue λ=100 en todas las ejecuciones de los problemas propuestos.

3.1 Problema de valor inicial (PVI)

Considere el siguiente problema de Cauchy tomado de (Tsoulos, 2006):

$$\begin{cases} y' = \dfrac{2x - y}{x} \\ y(0.1) = 20.1 \end{cases}$$

en el intervalo $x \in [0.1,1]$. La solución analítica está dada por la fórmula:

$$y(x) = x + \frac{2}{x}$$

Para buscar la solución de forma simbólica, la tabla 3.1 presenta los parámetros utilizados en el algoritmo de programación genética GP.

Tabla 3.1 Configuración del algoritmo GP para el problema de valor inicial.

Nodos	10 nodos uniformemente distribuidos en el intervalo $x \in [0.1,10]$
Operadores	Binarios: ×,/,+,- Unarios: *signo,cos,sin,tan,sqrt*
Parámetros del árbol solución	Máx. número de expresiones binarias en la solución 5. Máx. número de expresiones unarias en la solución 5. Máx. número de constantes 5.
Parámetros del AG	Tamaño población 500, Máx. núm. ciclos generacionales 100, tolerancia 10^{-7}, Mutación 0.1%, Cruce 0.9%, réplica 0.1%.

El código de la tabla 3.2 muestra las mejores soluciones obtenidas por el algoritmo de programación genética en uno de los experimentos ejecutados. La primera columna muestra la evaluación de adaptación del individuo y la segunda columna nos muestra su respectiva fórmula. En la tabla, el símbolo x0 se entenderá simplemente como la variable independiente x.

Tabla 3.2 Resultado de los mejores individuos durante una ejecución del problema de valor incial.

Valor función de adaptación	Fórmula
7074.321007831048,	(((-9 / -9) - (-2 / 5)) / x0)
2223.0,	(2 / x0)'
2191.974625187162,	((1 + (3 / (x0 + x0))) - (0 / (x0 + x0)))
591.9746251871594,	((x0 - -2) / x0)
591.9746251871594,	((2 + x0) / x0)
1.2190908517865785e-27,	(x0 + (2 / x0))

Los resultados de la tabla 3.2 muestran como el algoritmo de programación genética evoluciona para encontrar la respuesta exacta del problema. En este proceso evolutivo se evidencia la disminución del valor de la función de adaptación, como también la aparición emergente de un patrón en la estructura de los individuos la cual corresponde a una fórmula que resuelve el problema con la mejor función de adaptación que satisface la tolerancia establecida. En este caso particular al algoritmo le tomó 16 ciclos generacionales para encontrar la respuesta. Observe, adicionalmente, que la respuesta a pesar de ser única para el problema de valor inicial, no tiene una representación única como una fórmula en el espacio de búsqueda. Esto se puede apreciar en las diferentes fórmulas encontradas por el algoritmo tras ejecutar diferentes experimentos en la tabla 3.3. Note que todas tienen valor de adaptación del orden de 10^{-27} y se puede demostrar que son algebraicamente equivalentes.

Finalmente, se realizaron 100 experimentos con el fin de demostrar desde el punto de vista estadístico la convergencia del método para el problema 1. La Fig. 3.1 muestra la tabla de frecuencia acumulada en función del número de ciclos necesarios para obtener de forma exitosa una solución con una tolerancia de 10^{-7} en la función de adaptación. Como se puede apreciar, en más del 64% de los experimentos se obtuvo la respuesta exacta del problema en menos de 10 ciclos generacionales, demostrando la efectividad del método.

Tabla 3.3 Diferentes fórmulas de la solución del problema 1 obtenidas por el algoritmo de programación genética.

Valor función de adaptación	Fórmula
1.2190908517865785e-27,	(x0 + (2 / x0))
2.268137805072111e-27	(((x0 * x0) + 2) / x0)'
1.2190908517865785e-27	((((-3 - -6) - 1) * (1 / x0)) - ((-1 * x0)))
1.2190908517865785e-27,	(((2 / (1 * x0)) + x0) + (0 - 0))')
1.2190908517865785e-27,	(x0 - ((6 - 8) / x0))')
1.2190908517865785e-27,	((2 / x0) + (x0 * (1 * 1)))')

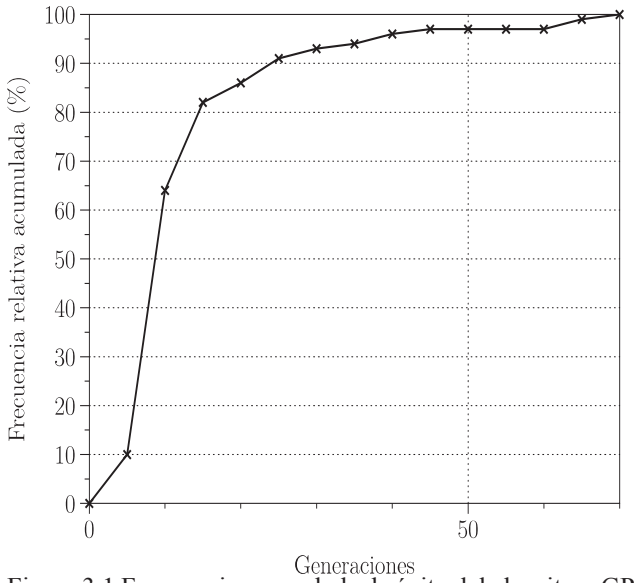

Figura 3.1 Frecuencia acumulada de éxito del algoritmo GP para el problema 1.

3.2 Problema no-lineal

Encontrar una función de una variable independiente y una variable dependiente, en forma simbólica que sea solución de la ecuación diferencial no-lineal:

$$\begin{cases} y'' = y^3 - yy', \\ y(1) = \frac{1}{2}, \\ y(2) = \frac{1}{3}, \end{cases}$$

en el intervalo $x \in [1,2]$. La solución analítica está dada por la fórmula:

$$y(x) = \frac{1}{x+1}$$

La tabla 3.4 describe la configuración del algoritmo de programación genética para este problema. La tabla 3.5 muestra la evolución de las mejores soluciones obtenidas por el algoritmo de programación genética en uno de los experimentos ejecutados del problema no lineal. Para este caso, al algoritmo le tomó 9 ciclos generacionales para encontrar una fórmula de la solución exacta del problema.

Tabla 3.4 Configuración del algoritmo GP para el problema no lineal.

Nodos	10 nodos uniformemente distribuidos en el intervalo $x \in$ **[1,2]**
Operadores	*Binarios: ×,/,+,- Unarios: signo.*
Parámetros del árbol solución	Máx. número de expresiones binarias en la solución 5. Máx. número de expresiones unarias en la solución 5. Máx. número de constantes 5.
Parámetros del AG	Tamaño población 500, Máx. Núm. ciclos generacionales 51, tolerancia 10^{-7}, Mutación 0.1%, Cruce 0.9%, réplica 0.1%.

Tabla 3.5 Evolución de los mejores individuos hacia una fórmula de la solución exacta durante una ejecución del problema no lineal.

Valor función de adaptación	Fórmula
3.7040729364283265	((4 - x0) / (6))
2.5816892016842443	(7 / (((4 + x0) * 4)))
1.8598086768615687	(- (2) / ((((x0 * -9) / (x0 * -9)) - x0) + -4))
1.8598086768615687	(- (2) / (((2 - 1) - x0) + -4))
0.4824414548709085	(- (2)/(((((x0*-9)/(x0*-9))*-9)/ (x0*-9)) -x0) +-4))
4.0444528832131953e-32	(- (2) / (((2 - x0) - x0) + -4))

La Fig. 3.2 muestra la tabla de frecuencia acumulada en función del número de ciclos necesarios para obtener de forma exitosa una solución con una tolerancia de 10^{-7} en la función de adaptación. Se puede apreciar nuevamente, que en más del 90% de los 65 experimentos realizados, se obtuvo la respuesta exacta del problema en menos de 10 ciclos generacionales.

3.3 Problema inestable (ill-conditioned)

El siguiente problema tiene por objetivo revisar el desempeño del método de programación genética cuando este se usa en problemas considerados en la literatura como numéricamente inestables. La mayoría de los métodos numéricos de un paso o multipaso presentan dificultades con este tipo de problemas, ver Mastorakis (Mastorakis, 2006).

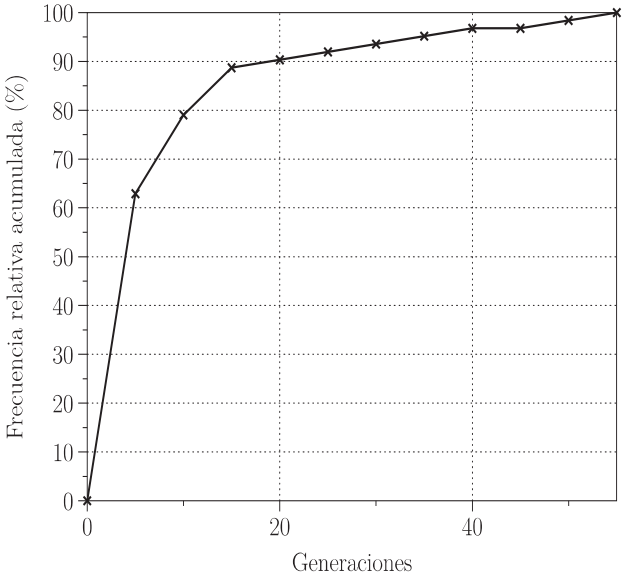

Figura 3.2 Frecuencia acumulada de éxito del algoritmo GP para el problema no lineal.

$$\begin{cases} y'' - 10y' - 11y = 0 \\ y(0) = 1 \\ y'(0) = -1 \end{cases}$$

en el intervalo $x \in [0,4]$. La solución analítica general es de la forma

$$y(x) = C_1 e^{11x} + C_2 e^{-x},$$

aplicando las condiciones iniciales obtenemos

$$y(x) = e^{-x}$$

La dificultad que presentan los métodos numéricos (un paso o multipaso) con este problema particular, consiste en que los errores acumulados propios del método numérico magnifican el término inestable e^{11x} de la solución, impidiendo de esta forma encontrar una aproximación satisfactoria para la condiciones iniciales dadas. Para ilustrar con más detalle esta situación, la figura 3.3 muestra la integración de la ecuación diferencial utilizando el método estándar *ode45* de Matlab en $x \in [0,4]$. La figura nos permite apreciar como la acumulación iterativa de errores de la rutina *ode45* se propagó a través del término inestable de la ecuación diferencial, impidiendo la convergencia en el intervalo.

Por su parte, el método de programación genética se ejecutó utilizando los parámetros de la tabla 3.6. La tabla 3.7 muestra la evolución de las mejores soluciones obtenidas en uno de los experimentos ejecutados del problema 3.3. Para este caso, el algoritmo encontró una fórmula de la solución exacta del problema en 5 ciclos generacionales.

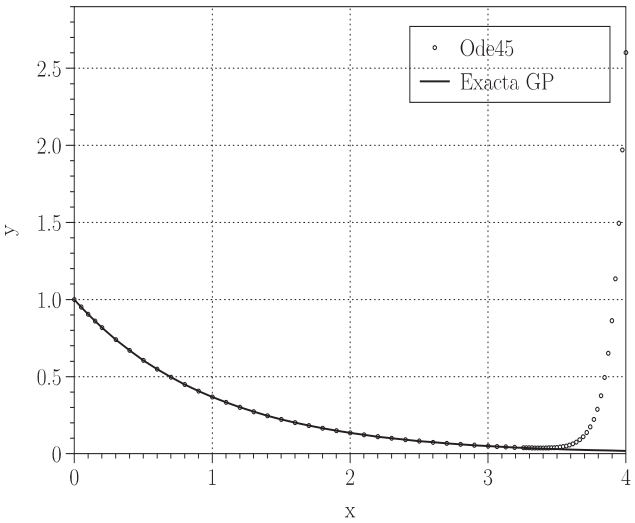

Figura 3.3 Comparación entre la solución aproximada obtenida con la rutina ode45 de Matlab y la solución exacta obtenida con el método de programación genética en el intervalo [0,4]. La acumulación iterativa de errores se propagan a través del término inestable de la ecuación diferencial, impidiendo la convergencia de la rutina ode45 en intervalos relativamente largos del dominio.

Finalmente, se realizaron 50 experimentos del problema inestable con el fin de mostrar estadísticamente la convergencia del método. La figura 3.4 muestra la tabla de frecuencia acumulada en función del número de ciclos necesarios para obtener de forma exitosa una solución con una tolerancia de 10^{-7} en la función de adaptación. Se puede apreciar, que más del 95% de los experimentos obtuvieron la respuesta exacta del problema en menos de 10 ciclos generacionales, demostrando la efectividad del método para el problema numéricamente inestable.

Tabla 3.6 Configuración del algoritmo GP para el problema inestable.

Nodos	40 nodos uniformemente distribuidos en el intervalo $x \in [0,4]$.
Operadores	*Binarios:* ×,/+,- *Unarios: signo -,exp,cos,sin.*
Parámetros del árbol solución	Máx. número de expresiones binarias en la solución 5. Máx. número de expresiones unarias en la solución 5. Máx. número de constantes 5.
Parámetros del AG	Tamaño población 500, Máx. núm. ciclos generacionales 200, tolerancia 10^{-7}, Mutación 0.1%, Cruce 0.9%, réplica 0.1%.

Tabla 3.7 Configuración del algoritmo GP para el problema inestable.

Valor función de adaptación	Fórmula
199.00972797180412	(exp (x0) / exp (x0))
199.00972797180412	(exp (-6) / exp (x0))
149.52901448310175	(exp ((-3 - x0)) * exp ((-3 - -4)))
47.76321486627132	((7 +(9 + -5))/exp((- (((7+-3) - (x0 + -1))) - -9)))
1.4239185858272146e-28	((-5 / -5) / exp (x0))

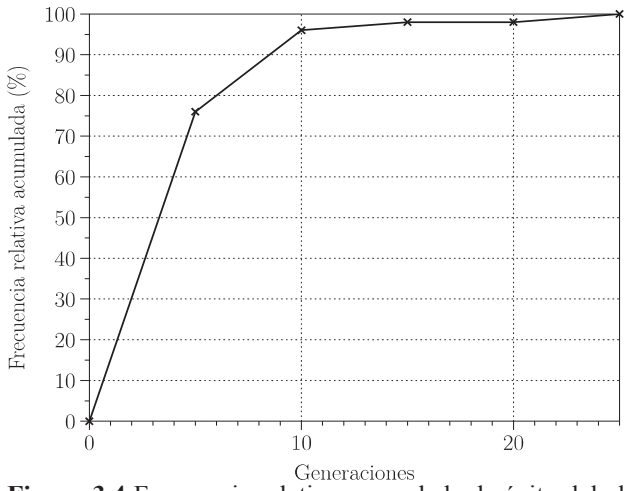

Figura 3.4 Frecuencia relativa acumulada de éxito del algoritmo GP para el problema inestable.

4. DISCUSIÓN

A partir de las secciones previamente desarrolladas este estudio permitió evidenciar las siguientes ventajas y desventajas en relación con el uso de un método basado en programación genética para hallar la solución simbólica de ecuaciones diferenciales.

4.1 Desventajas

- *Eficiencia computacional*. Es claro que los algoritmos genéticos son ineficientes desde el punto de vista del costo computacional para resolver ecuaciones diferenciales, cuando se le compara con los métodos numéricos convencionales y la programación genética no es la excepción. El número de evaluaciones funcionales que debe hacer el método para resolver una ecuación diferencial ordinaria puede ascender facilmente al orden de 10^6, una cifra un tanto desalentadora cuando se le compara con una técnica convencional como *ode45* cuyo número de evaluaciones funcionales para un problema de la misma naturaleza no asciende al orden de 10^4. Sin embargo, para problemas donde aumenta la complejidad computacional en el orden de la dimensionalidad, como por ejemplo métodos de elementos finitos para ecuaciones diferenciales parciales en tres dimensiones, la comparación del número de evaluaciones funcionales se podría reconsiderar, este es un tema para estudiar en el futuro.
- *Evaluación simbólica de las derivadas*. La evaluación simbólica de las derivadas, aunque garantiza la búsqueda de fórmulas que representan de forma exacta el problema, presenta una desventaja en relación con un aumento considerable en los tiempos de cómputo, ya que se hace necesario del uso de algoritmos recursivos sobre listas de operadores siguiendo las reglas de derivación establecidas.

4.2 Ventajas

Se puede ver que la principal desventaja del método se asocia al costo computacional, sin embargo, desde el punto de vista matemático y en aras de utilizar nuevas alternativas que permitan encontrar la solución de un problema, este es un precio razonable que se debe pagar. Si bien no hay *free lunch*, en la actualidad se están desarrollando diferentes técnicas que permiten reducir los costos de tiempo computacional como por ejemplo la computación en paralelo, la computación GPU, entre otros, la cuales se pueden implementar en el algoritmo a futuro. Por otra parte revisemos las ventajas que el método nos puede ofrecer.

- *Patrones emergentes.* El patrón emergente resultado de la recombinación genética de los individuos y de los operadores de réplica y mutación da como resultado una fórmula que representa la solución del problema. Aún cuando no se encuentre la solución exacta del problema por medio del algoritmo del programación genética, este patrón nos puede revelar el camino para la búsqueda de la solución de problema.
- *Versatilidad en la imposición de condiciones de frontera.* Una característica importante, consiste en la transparencia de la imposición de las condiciones de frontera en el problema, de esta forma toda condición de frontera simplemente se incluye como una penalidad en la función de adaptación tan solo como una diferencia al cuadrado. De esta forma se pueden incluir condiciones iniciales o de frontera como: condiciones ***Dirichlet***, condiciones de ***Neumann***, periódicas, o combinación de estas como las condiciones de ***Robin***.
- *Problemas no-lineales*. En general los algoritmos

genéticos son considerados como *last resource* en el sentido que se aplican para la solución de problemas difíciles que no se pueden resolver por medio de otros métodos de optimización convencionales. Estos problemas incluyen, problemas NP-Duros de optimización combinatorial, identificación de estructuras complejas, como también optimización de funciones multi-objetivo. Con respecto al problema no-lineal seleccionado, los autores reportan que los mismos no se pueden resolver con paquetes de cálculo simbólico convencionales como por ejemplo el paquete *Dsolve de Mathemática* (Tsoulosy Lagaris, 2006). Es aquí precisamente donde el uso de la programación genética puede jugar un papel importante como herramienta para resolver ecuaciones diferenciales.

- *Problemas inestables.* Otra característica a destacar consiste en la capacidad del método para resolver problemas inestables como el presentado en el Problema 11. Este sencillo pero ilustrativo ejemplo demuestra el potencial del método para resolver problemas aún más complejos que presentan dificultades que no pueden ser sorteadas por métodos convencionales.

4.3 Conclusiones

En este trabajo se desarrolló e implementó un método de programación genética para resolver ecuaciones diferenciales de forma simbólica partiendo del artículo de Tsoulos y Lagaris (Tsoulos,2006) y del libro fundamental de Koza (Koza,1992). Adicionalmente, se plantearon tres problemas eligidos de tal forma que permitieron evidenciar una serie de ventajas y de desventajas del uso del método en el campo de las ecuaciones diferenciales.

El método de programación genética se presenta como una alternativa importante y novedosa para la búsqueda de soluciones de ecuaciones diferenciales, si bien se requiere de un gran esfuerzo computacional comparado con los métodos tradicionales, este puede ser utilizado como último recurso para la búsqueda de soluciones particulares a problemas que pueden ser difíciles de implementar mediante otros métodos. Adicionalmente, desde el punto de la inteligencia artificial, en la resolución de problemas, el algoritmo puede ser utilizado en primera instancia como una herramienta para el desarrollo del bien denominado *Ansatz*, aquella intuición educada que nos permite encontrar la solución analítica de un problema.

AGRADECIMIENTOS

Los autores agradecen al grupo GEDES de la Universidad del Quindío y la Universidad Tecnológica de Pereira.

BIBLIOGRAFIA

1. D. Ashlock, Evolutionary computation for modeling and optimization, Springer Science+Business Media Inc., 2006.
2. R.M. Azif-Zahoor, J.A. Khan, and I.M. Qureshi, Evolutionary computation technique for solving ricatti differential equations of arbitrary order, World Academy of Science, Engineering and Technology 58 (2009), 303 – 309.
3. R. L. Burden, Analisis numérico, séptima edición ed., Cengage Learning, 2002.
4. R. De Castro, Notas de clase: Teoría de la computación, lenguajes, autómatas, gramáticas, Universidad Nacional de Colombia, 2004.
5. C. A. Coello-Coello, Introducción a la computación evolutiva, Notas de Curso CINVESTAV-IPN, Abril 2006.
6. J. M. Daida and A. M. Hilss, Identifying structural mechanisms in standard genetic programming, GECCO'03 (E. Cantu´-Paz et al., ed.), Springer-verlag Berlin Heidelberg 2003, 2003, pp. 1639– 1651.
7. D. Fagan, E. Hemberg, and M. O'Neill, Towards adaptive mutation in grammatical evolution, GECCO'12 July 7-11 2012, Philadelphia, PA, USA (ACM 9781-4503-1178-6/12/07, ed.), 2012, pp. 1481–1482.

8. M. A. Foster, The program structure of genetic programming trees, Master's thesis, School of Computer Science and Information Technology, RMIT University Melbourne Australia, October 2005.
9. J. García and J. Gutíerrez, Regresión Símbolica de funciones booleanas usando evolución gramatical, Tech. report, Universidad del Quind´ıo grupo GEDES, 2009.
10. J. H. Holland, Adaptation in natural and artificial systems, Ann Arbor The University of Michigan Press, 1975.
11. K. Jebari, M. Madiafi, and A. Moujahid, Solving poisson equation by genetic algorithms, International Journal of Computer Applications 83 (2013), no. 5, 1–6.
12. J. R. Koza, Genetic programming, 6th edition 1998 ed., Massachusetts institute of technology press, 1992.
13. N. E. Mastorakis, Unstable ordinary differential equations: Solution via genetic algortihms and the methos of nelder-mead, Proceedings of the 6th WSEAS int. Conf. on Sytems Theory and Scientific Computation Elounda Greece, August 2006, pp. 1–6.
14. M. O'Neill and C. Ryan, Grammatical evolution: Evolutionary automatic programming in an arbitrary language, Springer Science+Business Media Inc., 2003.
15. M. O'Neill, C. Ryan, M. Keijzer, and M. Cattolico, Crossover in grammatical evolution, Genetic Programming and Evolvable Machines 4 (2003), 67–93.
16. A. Quarteroni, R. Sacco, and S. Fausto, Numerical mathematics (text in applied mathematics; 37), Springer, 2000.
17. L. Schmitt and S. Droste, Convergence to global optima for genetic programming systems with dynamically scaled operators, GECCO'06, 2006, pp. 879–886.
18. Thorhauer and F. Rothlauf, Structural difficulty in grammatical evolution versus genetic programming, GECCO'13 July 6-10 2013, Amsterdam, The Netherlands, ACM, 2013, pp. 997–1004.
19. L. Tsoulos and I. E. Lagaris, Solving differential equations with genetic programming, Genetic Programming and Evolvable Machines 7 (2006), no. 1, 33–54.