Programación Genética y Evolución Gramatical: un enfoque combinado usando Straight Line Programs

Pablo Esaú Mejia Medina

Universidad de Cantabria

Este trabajo se presenta para obtener el grado de
Máster en Matemáticas y Computación

Junio 2019

Facultad de Ciencias

Programación Genética y Evolución Gramatical: un enfoque combinado usando Straight Line Programs

(Genetic Programming and Grammatical Evolution: A Combined Approach Using Straight Line Programs)

Trabajo de Fin de Máster
para acceder al

MÁSTER UNIVERSITARIO EN MATEMÁTICAS Y COMPUTACIÓN

Autor: Pablo Esaú Mejía Medina

Director: José Luis Montaña

Junio - 2019

Dedico estas memorias a mi madre Ercilia, a mi madrina Lourdes, a mis hermanos Kenia, Kevin, Eduardo y a mi abuela Ercilia . Por el apoyo, consejos, paciencia y motivación brindados a lo largo de mi vida y particularmente en este trabajo.

Declaración

Por la presente declaro que el trabajo descrito en esta disertación es, salvo que se indique lo contrario, totalmente de mi propio trabajo y no se ha presentado como un ejercicio para obtener un título en esta o cualquier otra universidad.

Pablo Esaú Mejia Medina
Junio 2019

Agradecimientos

En primer lugar, desearía agradecer profundamente a mis dos madres, Ercilia Medina y Lourdes Vásquez, por el apoyo incondicional y motivación que me han brindado a lo largo de mi vida. Sin él, posiblemente hubiera abandonado ante las primeras dificultades.

Quiero mostrar mi gratitud, tanto a mis amigos como compañeros de estudio, particularmente a Idalia Romero. Agradezco a mi asesor Luis Montaña, por las correcciones observadas.

Este trabajo está escrito en gran medida con software libre, en particular, quiero agradecer a los participantes de la comunidad de: Latex, GNU, Linux, c/c++ y python.

No puedo terminar estas líneas sin agradecer a la banda de rock dream theater, cuyas piezas me acompañaron a lo largo de este trabajo.

Abstract

English

One of the main problems in genetic programming for solving symbolic regression problem is to design the topology of the data structure, that is, to determine the number of suitable operations (not less, not too many) required to solve the problem. In the present work a new algorithm based on genetic programming, straight line programs and grammatical evolution is adopted. As a result, it produces simpler Straight Line Programs (SLPs) that have a better generalization capacity and is easier to implement. Moreover, due to the fact that the generalization capacity of SLPs may decrease, the algorithm uses a novel adaptive penalty approach to adjust the structure generated through the process of evolution. The proposed method has been tested with two sets of target functions previosly used in the literature and the results are contrasted with other similar algorithms that use Straight Line Programs as data structure in genetic programming. The experimental results suggest that our method presents improvements in most cases and that it is a reasonable approach to consider for solving symbolic regression problem.

Español

Uno de los principales inconvenientes en programación genética (GP) para resolver el problema de la regresión simbólica es diseñar una topología de la estructura de datos utilizada, esto es, determinar el número de operaciones adecuadas (no menos, ni demasiadas) que se requiere para solucionar el problema. En el presente trabajo se adopta un nuevo algoritmo basado en programación genética, straight line programs (SLPs) y evolución gramatical para diseñar la topología de los SLPs. Como resultado, produce SLPs más simples que tienen una mejor capacidad de generalización y son más fáciles de implementar. Además, debido al hecho de que la capacidad de generalización de los SLPs puede disminuir, el algoritmo

utiliza un novedoso enfoque de penalización adaptativa para ajustar los SLPs generados a través del proceso de evolución. El método propuesto ha sido analizado con dos conjuntos de funciones objetivo previamente usadas en la literatura y los resultados han sido contrastados con otros algoritmos similares que usan SLPs como estructura de datos en GP. Los resultados experimentales sugieren que nuestro método presenta mejoras en la mayoría de los casos y que es un enfoque razonable a considerar para resolver regresión simbólica.

Índice de contenido

Índice de figuras

Índice de tablas

Lista de definiciones

Capítulo 1

Introducción

La programación genética (GP) es una variante de los algoritmos genéticos, que genera automáticamente expresiones simbólicas (S-expresión), en particular programas de computadora que realizan una tarea específica ([6], [5]). El problema de inducir programas puede ser reformulado tal como se explica en [13] como una búsqueda en el espacio de todos los programas. Este espacio es determinado por un conjunto de terminales y funciones. Bastaría entonces, con elegir un conjunto de terminales y funciones adecuadas para solucionar un determinado problema. El problema que intentamos resolver a lo largo de este trabajo es conocido en la literatura como **regresión simbólica** [12]. Particularmente se aborda desde el enfoque de programación genética usando como estructura de datos Straight Line Program (SLP) combinado con Evolución Gramatical (GE).

Los circuitos aritméticos son conocidos en la literatura como un tipo particular de SLP, que tomando las operaciones aritméticas estándar $\{+, -, *, /\}$ han sido usados como modelos de computación para estudiar la complejidad computacional de algoritmos en muchos campos de la matemática, por ejemplo: en computación algebraica, en problemas de álgebra línea, en teoría de eliminación y en geometría algebraica (ver [10] y [8] para profundizar en estos temas). En años recientes ha surgido un enfoque que usa como estructura de datos Straight Line Program (SLP) para representar programas en el paradigma lineal de programación genética.

Los Straight Line Programs son usados por primera vez en programación genética en [4]. Este trabajo muestra las ventajas de usar esta estructura de datos ante la tradicional estructura de árboles. Usar Straight Line Program reduce en gran medida la cantidad de operaciones que se necesitan para conseguir una *buena* aproximación a los datos tratados en el problema

de regresión simbólica.

El número de instrucciones, longitud, altura, arquitectura o número de operaciones de un SLP tiene una enorme importancia en la capacidad de aprendizaje y generalización de los mismos en programación genética. En este trabajo se presentan resultados experimentales obtenidos al hacer pruebas de nuestro enfoque en programación lineal genética, basado en SLP y evolución gramatical sobre el problema de regresión simbólica y se comparan los resultados obtenidos sobre el mismo problema por enfoques similares, es decir, que usan SLP como estructura de datos en programación genética.

Imaginamos nuestro desarrollo como la implementación más simple posible impulsada por una función fitness y un método de penalización para diseñar la arquitectura de los SLPs que refleje su capacidad para resolver el problema considerado. En este sentido, los resultados descritos en este trabajo son solo un paso natural hacia un escenario de GP en el que la estructura SLP se utiliza para resolver problemas del mundo real.

El trabajo se organiza de la siguiente manera:

- En el capítulo 2 se explican algunos elementos importantes de la programación genética, la estructura de datos *Straight Line Program*, la evolución gramatical y el problema de la regresión simbólica, además, algunas propiedades, aspectos y conceptos relacionados.
- El capítulo 3 describe la formulación del problema.
- En el capítulo 4 se presenta el algoritmo propuesto y se describe en detalle el mismo, haciendo énfasis en los operadores genéticos utilizados y además se explica el método de penalización propuesto.
- En el capítulo 5 se describe como se realizó la experimentación y el conjunto de funciones objetivo, consideradas para los experimentos y finalmente,
- El capítulo 6 muestra los resultados experimentales. Se extraen conclusiones y futuras direcciones de investigación.

Capítulo 2

Estado del arte

Para explorar de manera más efectiva los aspectos más importantes relacionados con la programación genética (GP), el problema de regresión simbólica, los SLPs en GP y la evolución gramatical, se seleccionó un subconjunto de trabajos de la literatura según su relevancia para contestar las siguientes preguntas:

1. ¿ Cómo se ha usado la programación genética (GP) en el problema de regresión simbólica?
2. ¿ Cómo los SLPs han sido usados en GP?
3. ¿ Qué es la evolución gramatical y cómo se ha usado en GP?

2.1 Programación genética y regresión simbólica

Con el fin de responder a la pregunta uno, se describen dos componentes principales, la primera: descripción general del paradigma de la programación genética y la segunda: descripción detallada de la programación genética.

Visión general del paradigma de programación genética

Los *algoritmos genéticos* (GA) están inspirados en un proceso darwiniano [1]. Este es un tipo de algoritmo matemático que transforma un conjunto de objetos matemáticos individuales

[1] Selección natural de pequeñas variaciones heredadas que aumentan la capacidad del individuo para competir, sobrevivir y reproducirse.

(población). Cada uno está asociado con un valor que mide su fitness o rango en la población actual, este valor se conoce como función de fitness. Para la reproducción y la supervivencia se aplican operaciones con patrones, estas operaciones se conocen como cruce y mutación. El paradigma de ***programación genética*** descrito en esta sección aplica muchas de las ideas clave del algoritmo genético general.

La programación genética (GP) comienza con una población inicial generada aleatoriamente, en este caso, los objetos matemáticos son programas de computadora compuestos de funciones y terminales apropiadas para el dominio del problema. Según [5] *muchos problemas aparentemente diferentes en inteligencia artificial, procesamiento simbólico y aprendizaje automático pueden considerarse como un descubrimiento de un programa de computadora que produce algunos resultados deseados para entradas particulares*, y como se explica en [13] el proceso de solución de estos problemas se puede reformular como una búsqueda de un programa altamente adecuado en el espacio de posibles programas de computadora. Como en los algoritmos genéticos (GA) en programación genética (GP), cada individuo (programa de computadora) en la población se mide por el valor de fitness, esto es, lo bien que funciona en el entorno del problema particular. La forma de medir el fitness de un individuo o dicho de otra forma la manera de definir la función de fitness varía con el problema.

Para mejorar a los individuos en la población actual es necesario algún mecanismo que permita generar nuevos individuos (descendencia) con algunas características de los antiguos, este mecanismo se conoce como proceso de evolución, este proceso comienza con la operación genética de la recombinación sexual (cruce o crossover). La operación de reproducción implica seleccionar, de acuerdo con su valor fitness u otros mecanismos de selección, un programa de computadora de la población actual de programas (llamados padres), y permitir que sobreviva al copiarlo en la nueva población. Los programas para padres no necesariamente tienen los mismos tamaños y formas. Los programas de descendencia están compuestos por sub-expresiones (sub-árboles, sub-gráficos, sub-programas, bloques de construcción, sub-rutinas, etc.) de sus padres. Estos programas de descendencia son típicamente diferentes a sus padres. Después de que las operaciones de reproducción y cruce se realizan en la población de descendientes (la nueva generación) reemplaza a la población antigua (la generación anterior). El proceso se repite a lo largo de muchas generaciones.

En conclusión, GP genera programas de computadora (**programas** desde ahora en adelante), para resolver problemas ejecutando los siguientes tres pasos:

1. Generar una población aleatoria inicial de programas.

2. Realizar de forma iterativa los siguientes subpasos hasta que se cumpla el criterio de finalización:

   (a) Para cada programa se calcula el valor de fitness.

      i. Copiar los programas existentes a la nueva población.

      ii. Crear nuevos programas mediante la recombinación genética de partes elegidas al azar de dos programas existentes.

3. Como resultado de la programación genética, se designa el mejor programa que apareció en cualquier generación. Este resultado puede ser una solución (o una solución aproximada) al problema.

Descripción detallada de la programación genética

Definición 1 (Conjunto de posibles estructuras en GP.)**.** *Sean $F = \{f_1, ..., f_n\}$ el conjunto de funciones y $T = \{t_1, ..., t_m\}$ el conjunto de terminales. Sea $a_i = a(f_i)$ la aridad para $f_i \in F$. Entonces, el conjunto de las posibles estructuras en GP es el conjunto de todas las posibles combinaciones de F y T sintácticamente correctas.*

El conjunto de funciones puede incluir una clase diversa de funciones, por ejemplo:

1. Funciones básicas como las operaciones aritméticas (+,-,*,etc.),
2. Funciones matemáticas tales como $\sin, \cos, exp, log, sqrt$, etc.,
3. Operaciones Booleanas,
4. Operadores condicionales,
5. Funciones que causan iteración, y
6. cualquier otra función específica de dominio que pueda definirse.

Las terminales son típicamente variables atómicas o constantes atómicas.

Propiedad de suficiencia

Una característica importante en GP que involucra funciones y constantes es la propiedad de suficiencia. Esta propiedad es una forma natural de exigir que el conjunto de terminales (T) y el conjunto de funciones primitivas (F) sean capaces, a través de combinaciones o configuraciones específicas, de expresar una solución al problema [5].

2.1.1 Regresión simbólica

La regresión simbólica, en términos generales, consiste en evolucionar una expresión simbólica que se ajusta de la manera más precisa posible a una muestra de datos finita dada. Es un caso particular del problema de regresión. De manera más formal y, en consecuencia, con el trabajo en [4], se da la siguiente definición. En lo sucesivo $\mathbb{R}$ representa el conjunto de los números reales.

Definición 2 (Regresión Simbólica)**.** *Sea $X = \mathbb{R}^n$, $Y = \mathbb{R}$ y $z = (x_i, y_i)_{1 \leq i \leq m} \in (X \times Y)^m$ los datos (que consisten de un conjunto de puntos) que se distribuyen según una medida de probabilidad desconocida ρ en $Z = X \times Y$ y son independientes e idénticamente distribuidos $(i.i.d)$. Se requiere construir una función $f : X \to Y$ en forma simbólica que prediga el valor $y \in Y$ a partir de un $x \in X$ dado, minimizando el error empírico sobre la muestra. El error empírico de la función f es:*

$$\varepsilon_z(f) = \frac{1}{m} \sum_{i=1}^{m} (f(x_i) - y_i)^2 \tag{2.1}$$

conocido como el error cuadrático medio o mean square error (MSE).

De acuerdo con la definición anterior y la descripción anterior de GP, el problema de regresión simbólica se puede representar mediante la siguiente idea:

1. Objetivo: encontrar una función en forma simbólica, que se ajuste a una muestra dada de puntos de datos z, donde z es como en la definición 2.
2. Conjunto de terminales: variables independientes y constantes.
3. Conjunto de funciones: $\{+, -, *, \div, \sin, \cos, exp, rlog, etc\}$
4. Función de fitness: para cada programa se calcula el valor de la ecuación 2.1

Por lo general, estos programas se han codificado utilizando árboles dirigidos con ramas ordenas o usando Programación Genética Lineal (LGP). En los últimos años se presentó una nueva estructura de datos para representar programas en el paradigma de programación genética lineal (LGP) : straight line programs (SLPs) en [4], que muestra una mejora considerable en el rendimiento comparado a la representación mediante árboles.

2.2 Uso de los Straight Line Programs en Programación Genética

Como se explicó en la sección 2.1, los Straight Line Programs se han utilizado en muchos campos de la matemática y la teoría de la complejidad, pero en GP se usó por primera vez en [4]. Donde se describió una nueva forma de hacer cruce entre SLPs. Para responder a esta pregunta, hacemos una breve descripción de los aspectos formales de esta estructura de datos, guiados por el trabajo anteriormente mencionado.

2.2.1 Estructura de datos Straight Line Program

Definición 3 (Estructura de datos Straight Line Program)**.** *Sean F y T como en la definición 1. Un Straight Line Program (SLP) sobre F y T es una secuencia finita de instrucciones computacionales* $\Gamma = \{I_1, ..., I_l\}$ *donde:*

$$I_k \equiv u_k := f_{jk}(\alpha_1, ..., \alpha_{a_{jk}});\ conf_{jk} \in F$$

$$\alpha_i \in T \text{ para todo } i \text{ si } k = 1 \text{ y } \alpha_i \in T \cup \{u_1, ..., u_{k-1}\} \text{ para } 1 < k \leq l$$

La longitud (l) de un SLP es determinada por el número de instrucciones de Γ.

Observación: *el SLP* Γ *es el código del programa, entonces para cada instrucción* $I_i \in \Gamma$ *una nueva variable* u_i *es introducida*. De lo anterior se puede establecer $\Gamma = \{I_1, ..., I_l\}$ por $\Gamma = \{u_1, .., u_l\}$ y $|\Gamma| = l$. Las variables u_i son expresiones sobre el conjunto de terminales, pero $u_i \notin T$ (son no terminales) y son construidas por una secuencia de composiciones recursivas desde el conjunto de funciones F. El conjunto de todos los SLPs sobre F y T será denotado por $SLP(F,T)$.

Ejemplo 2.2.1. *Sean* $F = \{+, -, *\}$ *y* $T = \{1, x_1, x_2\}$ *. Entonces SLP(F,T) es una secuencia de polinomios en 2 variables con coeficientes en los enteros. Considerando el SLP definido en 2.2 con longitud* $l = 5$*, se genera el polinomio* $u_5 = 2x_1^2x_2 - x_1^2$

$$\Gamma \equiv \begin{cases} u_1 & := x_1 * x_1 \\ u_2 & := u_1 + u_1 \\ u_3 & := u_1 * x_2 \\ u_4 & := u_3 - u_1 \\ u_5 & := u_4 + u_3 \end{cases} \tag{2.2}$$

Por otro lado, es importante tener una forma de identificar cuando dos SLPs son equivalentes. Esto último se da cuando los SLPs tienen la misma función semántica. La función semántica es un mapeo desde el conjunto de instrucciones al conjunto de salida asociado al SLP, más formalmente:

Definición 4 (Conjunto de Salida)**.** *Sea* $\Gamma = \{u_1, ..., u_l\}$ *un SLP sobre F y T. Un conjunto de salida de* Γ*,* $O(\Gamma) = \{u_{i1}, ..., u_{it}\}$*, es cualquier conjunto de variables no terminales de* Γ*.*

Definición 5 (Función semántica)**.** *Sea* $V = \{x_1, ..., x_p\} \subset T$ *el conjunto de variables terminales, la función semántica de* Γ*, denotado como* $\Phi_\Gamma : I^p \to O^t$*, satisface* $\Phi_\Gamma(a_1, ..., a_p) = (b_1, ..., b_t)$*, donde* b_j *representa el valor de la expresión sobre V de la variable no terminal* u_{ij} *cuando se sustituyen las variables* x_k *por* a_k*;* $1 \leq k \leq p$*.*

En este trabajo el conjunto de salida será constituida por una sola variable.

2.2.2 Código efectivo y no efectivo

El concepto sobre código efectivo y no efectivo en SLP es de gran importancia porque nos permite saber qué parte del código es necesario para calcular el resultado para una instrucción $u_k \in \Gamma$ y cuál no. Particularmente, el código efectivo se refiere al conjunto de instrucciones involucrado en los cálculos de la función semántica asociada al SLP, por ejemplo, y sin pérdida de generalidad, en el ejemplo anterior 2.2 y considerando como $O(\Gamma) = \{u_5\}$, nótese que para calcular $\Phi(\Gamma)$ para una entrada $(a_1, a_2) \in \mathbb{R}^2$, esto es, $\Phi_\Gamma(a_1, a_2)$, no es necesario calcular el valor intermedio de la instrucción u_2 en Γ. En este caso la instrucción u_2 es considerada como código no efectivo y deberá ser removida, renombrando adecuadamente las piezas de código efectivo, entonces el nuevo SLP Γ' que se obtiene es:

$$\Gamma' \equiv \begin{cases} u_1 & := x_1 * x_1 \\ u_2 & := u_1 * x_2 \\ u_3 & := u_2 - u_1 \\ u_4 & := u_3 + u_2 \end{cases} \tag{2.3}$$

Los SLPs Γ y Γ' son equivalentes ya que tienen la misma función semántica, esto es $\Phi_\Gamma(x, y) = 2x^2y - x^2 = \Phi_{\Gamma'}(x, y)$.

Para el cálculo del código efectivo de un SLP se estableció una relación en el conjunto de las variables no terminales. Esencialmente, una instrucción u_i está relacionada con otras instrucciones u_k si se necesita u_i para calcular u_k. Más formalmente:

Definición 6 (Relación entre instrucciones)**.** *Sea* $\Gamma = \{u_1, ..., u_l\}$ *un SLP sobre F y T. Asuma* $u_i := f(\alpha_1, ..., \alpha_{a_{ji}})$ *y* $u_k := f(\beta_1, ..., \beta_{a_{jk}})$*, con* $i < k$*. Entonces* u_i *está relacionada con* u_k*, esto es* $u_i R u_k$ *si y solo si* $u_i = \beta_s$*, para algún* $s, 1 \leq s \leq a_{jk}$

Observación: *Si se considera* $\hat{R}$ *como la clausura reflexiva y transitiva de R, entonces constituye una relación de equivalencia sobre* Γ

2.2.3 Straight Line Programs en Programación Genética

Teniendo en cuenta el componente principal en GP, estas subsecciones describen cómo generar la población inicial, la función de fitness y los operadores de recombinación utilizados en [4] (vea este documento para una comprensión completa de estos temas).

Población Inicial

La generación de cada SLP en la población inicial se realiza de la siguiente manera:

1. Primera instrucción $k = 1$:
   (a) Para generar la primera instrucción u_1 tenemos que seleccionar una función de F aleatoria, es decir, $f_{j1} \in F$
   (b) Para cada argumento $i \in \{1, ..., a_{j1}\}$ de f_{j1} elegir aleatoriamente $\alpha_j \in T$
2. Para $k > 1$:
   (a) Se elige de manera aleatoria una función $f_{jk} \in F$.
   (b) Para $i = \{1, ..., a_{jk}\}$ un elemento $\alpha_i \in T \cup \{u_1, ..., u_{k-1}\}$ es elegido de manera aleatoria.

Observación: *Los SLPs generados con la estrategia anterior podrían tener partes de código no efectivas*, pero no importa, ya que el objetivo es permitir código no efectivo durante el proceso de evolución.

Función fitness

La función fitness es el resultado de aplicar la fórmula de la definición 2.1 a la función semántica asociada al SLP, esto es:

$$MSE_z(\Gamma) = \varepsilon_z(\Phi_\Gamma) = \frac{1}{m}\sum_{i=1}^{m}(\Phi_\Gamma(x_i) - y_i)^2 \tag{2.4}$$

Operadores de Recombinación

Como se explicó en la subsección 2.1 en GP, se requieren dos operadores principales, la mutación y el cruce.

Definición 7 (Cruce (Crossover))**.** *Sean* $\Gamma = \{u_1, ..., u_L\}$ *y* $\Gamma' = \{u'_{j1}..., u'_L\}$ *dos SLPs sobre F y T. Sea* $u_k \in \Gamma$ *con k seleccionado aleatoriamente de* $1 \leq k \leq L$*. Sea* S_{u_k} *el conjunto de piezas de código efectivo de* Γ *relacionado para la evaluación de* u_k*, esto es:*

$$S_{u_k} = \{u_j \in \Gamma / u_j \hat{R} u_k\} = \{u_{jq}, ..., u_{jm}\}; con j1 <, ..., < jm \tag{2.5}$$

Entonces, el operador de cruce consiste en los siguientes pasos:

1. *Seleccionar* $u_t \in \Gamma'$ *aleatoriamente de* $m \leq t \leq L$
2. *Modificar* Γ' *sustituyendo el conjunto de instrucciones* $\{u'_{t-m+1}, ..., u'_t\}$ *en* Γ' *por las instrucciones en* S_{uk} *adecuadamente renombradas. Para el renombramiento se define una función* $\mathscr{R}$ *sobre* S_{uk} *por* $\mathscr{R}(u_{ji}) = u'_{t-m+i}$ *para* $i \in \{1, ..., m\}$*. Lo anterior genera la primera descendencia.*
3. *Para la segunda descendencia se repite el proceso con el otro SLP, esto es* Γ'

2.3 Evolución Gramatical en Programación Genética

La Evolución gramatical (GE) es un enfoque de la programación genética. GE tiene la propiedad de generar un sistema altamente flexible y fácil de usar, además, nos permite explotar una rica modularidad en sus diseños. La evolución gramatical emplea genomas lineales, realiza un mapeo ontogenético desde el genotipo hasta el fenotipo (programa) y utiliza una gramática para introducir estructuras legales en el espacio fenotípico, esos tres aspectos difieren la GE del GP tradicional [9].

La gramática proporciona un mecanismo único para usar en GP, esta describe muchas estructuras complejas. Evolucionar una estructura directamente puede atraer problemas como usar un operador genético no válido (cruce) y quizás lo más importante, generar

individuos inviables, esta es otra ventaja en la evolución gramatical, ya que la búsqueda se realiza en la gramática (esto es en la secuencia requerida para producir la estructura deseada). Este componente hace de la gramática una gran herramienta como parte de la computación evolutiva.

2.3.1 Notación de la gramática: Backus Naur Form

La gramática tiene que ser expresada en alguna forma, la notación más común para expresarlas es a través de reglas de producción, esta notación se conoce como Bakus Naur Form (BNF). Es bien conocido en la literatura que las gramáticas de BNF consisten en terminales, que en términos generales son elementos (por ejemplo, +, -, etc.) y las reglas de producción (no terminales). Una gramática BNF se define por una 4-tupla {N, T, P, S }, donde:

- N: denota el conjunto de símbolos no terminales.
- T: conjunto de símbolos terminales
- P: reglas de producción.
- S: símbolo inicial (comienzo del proceso).

2.3.2 EL papel de la Evolución Gramatical usada en Programación Genética

En programación genética (GP) se ha utilizado la Evolución Gramatical en muchos y diferentes aspectos. LOGENPRO es un sistema basado en LOgic grammar GENetic PROgramming(programación genética usando gramática lógica) y fue desarrollado por Wong y Leung. Para representar los individuos (programas) en LOGENPRO se utiliza una estructura de datos de árbol. Otro enfoque es la programación genética binaria (BGP) basada en la programación genética lineal que emplea un mapeo genotipo-fenotipo (es decir, un mapeo del genotipo al fenotipo). BGP utiliza una gramática libre de contexto (CFG) durante el mapeo. El algoritmo genético para software derivado (GADS) es otro enfoque que utiliza BGP como un proceso de mapeo genotipo-fenotipo para generar individuos, en este caso, las gramáticas de BNF se extienden para incluir un símbolo predeterminado para cada no terminal. El objetivo es el uso de un símbolo predeterminado cuando el proceso de asignación está incompleto, es decir, los elementos del conjunto no terminal siguen apareciendo en la expresión que ha sido mapeado. CFG / GP (Context Free Grammar GP) es otro enfoque y es muy similar a GADS.

Sin embargo, la diferencia es aplicar las reglas especificadas por cada gen al siguiente no terminal, cada gen se lee y se puede aplicar a cualquier no terminal adecuado. (Se sugiere leer [14], [12], [7] para ampliar sobre estos temas).

Capítulo 3

Planteamiento del problema

Hasta ahora, se ha presentado como la combinación de los Straight Line Program con programación genética es un buen enfoque en el problema de regresión simbólica, sin embargo, este presenta un inconveniente que se describe a continuación.

3.1 La longitud de los Straight Line Programs

Para entender este problema, damos una ilustración. Supongamos que se requiere aproximar los datos $Z = \{(x_i, y_i)\}_{1 \leq i \leq m}$ generados acorde a $f = x^5 + x^4 + x^3 + x^2 + x + 1$. Para resolver esto, se puede pensar en $F = \{+, -, *, /\}$ y $T = \{x, 1\}$. Ahora consideremos el siguiente SLP con longitud máxima $l = 3$, entonces, f puede ser aproximada mediante la siguiente secuencia de instrucciones:

$$\Gamma = \begin{cases} u_1 & := x * x \\ u_2 & := u_1 * u_1 \\ u_3 & := u_2 * x \end{cases} \tag{3.1}$$

Nótese que, $u_3 = x^5$ no se ajustará de la mejor manera a Z. Por otro lado, si consideramos a Γ con $l = 4$, se puede generar la siguiente secuencia de instrucciones.

$$\Gamma = \begin{cases} u_1 & := x * x \\ u_2 & := u_1 * u_1 \\ u_3 & := x + 1 \\ u_4 & := u_3 * u_2 \end{cases} \tag{3.2}$$

Dado que $u_4 = x^5 + x^4$, es claro que este conjunto de instrucciones se ajustará un poco mejor a los datos que el anterior, pero igualmente seguirá teniendo un error. Ahora considere $l = 6$

$$\Gamma = \begin{cases} u_1 & := x * x \\ u_2 & := u_1 * u_1 \\ u_3 & := x + 1 \\ u_4 & := u_3 * u_1 \\ u_5 & := u_3 * u_2 \\ u_6 & := u_5 + u_4 \end{cases} \tag{3.3}$$

Se obtiene, $u_6 = x^5 + x^4 + x^3 + x^2$ que se ajusta de una mejor forma que los anteriores. Lo anterior nos indica que el número máximo de operaciones o altura de los SLP puede tener un papel importante en el problema de regresión simbólica, ya que elegir una altura máxima que sea corta para ajustarse al problema puede causar una pobre solución (como con $l = 3,\ 4$ en el ejemplo anterior) o en su defecto no ser una solución, y establecer una altura máxima muy por encima de la necesaria provocará mucho código no efectivo. El algoritmo presentado en [4] asume que conocemos la longitud máxima del SLP, **pero en un problema real, en general, no tenemos manera de conocer de antemano la forma de la función objetivo y consecuentemente la longitud del SLP**. Este trabajo presenta un algoritmo evolutivo que nos permite aproximar los datos (Z) utilizando SLP sin conocer de antemano el número de operaciones requeridas. A continuación se explica tal algoritmo.

Capítulo 4

Algoritmo propuesto

Grammatical Straight Line Program Genetic Programming (GSLPGP) es un algoritmo basado en la programación genética y la evolución gramatical, que genera como salida SLPs. El procedimiento que incluye la evolución gramatical se conocerá como la parte topológica. La estructura general del algoritmo se muestra en la figura 4.1. La representación de la parte topológica, los operadores genéticos y el método de penalización propuesto se describen en detalle en la sección 4.2.

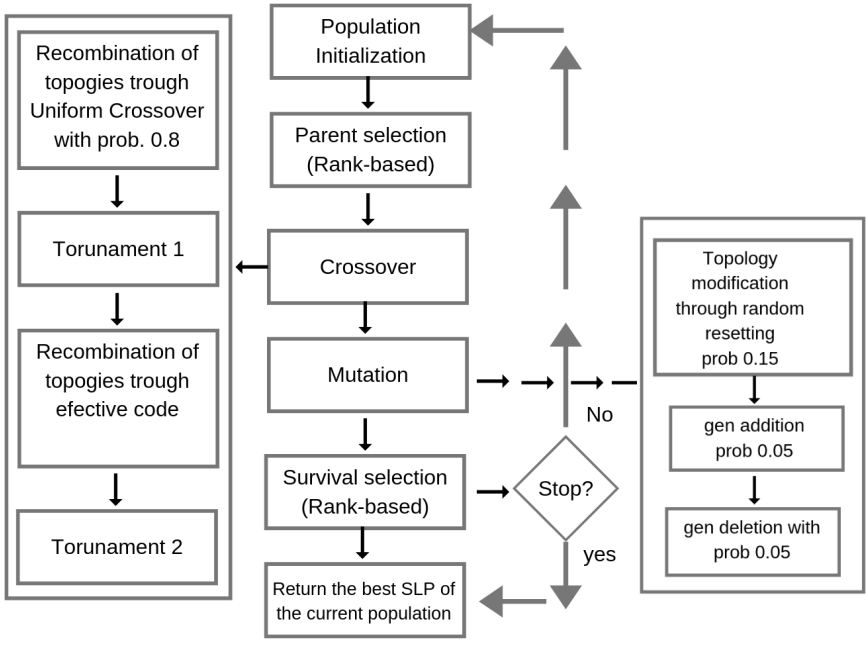

Fig. 4.1 Estructura del algoritmo propuesto

4.1 Representación de la parte topológica

Cada cromosoma incluye una parte topológica y se construye con una cantidad diferente de genes. Cada gen es codificado con un número en el rango de [0,255]. En la población inicial se establece el número máximo de genes para cada cromosoma mediante la siguiente fórmula:

$$\text{gen\_number} = 4 * l, \tag{4.1}$$

donde l es la longitud del SLP y $1 \leq l \leq h_{\max}$ con $h_{\max}$ como el número máximo de instrucciones para los SLPs o longitud máxima. Algunas características de los SLP están predeterminadas en la parte de topológica.

Predecesor	:=	Producciones	Índice de la producción
<S>	:=	<T><op><T> | <I><op><T> | <T><op><I> | <I><op><I>	0 1 2 3
<op>	:=	+ |- |* |/	0 1 2 3
<T>	:=	t_1 | t_2 | $\vdots$ | $\vdots$ |t_n	0 | 1 | $\vdots$ | $\vdots$ n-1
<I>	:=	u_1 | u_2 | $\vdots$ | $\vdots$ |u_l	0 | 1 | $\vdots$ | $\vdots$ l-1

Table 4.1 Gramática libre de contexto propuesta

La gramática propuesta es libre de contexto (CFG) y está representada en BNF. La CFG se muestra en la tabla 4.1, donde $t_1, ..., t_n$ son constantes o variables, $u_1, ..., u_l$ son las

instrucciones generadas (no terminales) por las producciones $< S >$. Para dar una ilustración de lo que eso significa, veamos el siguiente caso.

Ejemplo 4.1.1. *Para aproximar* $f = x^3 + x^2 - x + 1$*, considere {0,1} como constantes y variable x, esto es,* $F = \{+, -, *, /\}$ *y* $T = \{x, 0, 1\}$

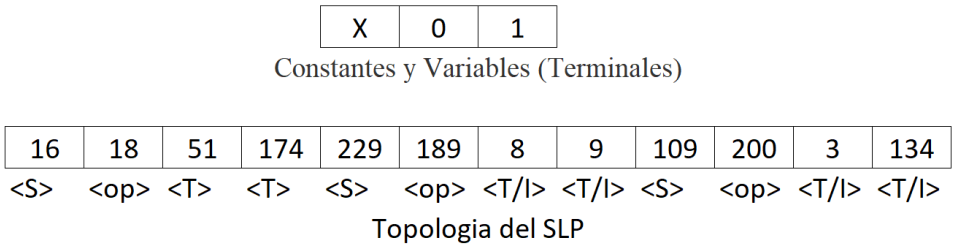

Fig. 4.2 En la parte superior la representación de las constantes y en la parte inferior la representación de los cromosomas.

Para dar solución al ejemplo anterior, primero se muestra en la figura 4.2 la representación cromosómica y cómo manejar las constantes. En segundo lugar, debe observarse que los genes se asignan con cada símbolo de la gramática propuesta (ver tabla 4.1). El primer gen corresponde al símbolo $< S >$, el segundo a $< op >$, el tercero y cuarto corresponden a los argumentos (en este caso terminales o instrucciones). Los pasos del proceso de decodificación se presentan en la figura 4.3 y se describe en los siguientes pasos:

1. El proceso comienza con el símbolo inicial $< S >$. Para generar un SLP correcto, debemos garantizar que el valor del primer gen sea un múltiplo de cuatro, porque la primera instrucción (I) debe ser generada por la producción $< T >< op >< T >$, por lo tanto, el valor del primer gen módulo 4 es 0, los demás genes se configuran con números aleatorios entre $[0, 255]$. Para descodificar la primera proyección seleccionada ($< T >< op >< T >$) continuamos con los siguientes pasos:

    - El siguiente gen correspondiente a la expresión <op>, tiene el valor 18 y tenemos 4 operaciones, entonces, 18 mod 4 es 2, que corresponde a la operación multiplicación (*).
    - El siguiente gen es el primer operando (<T>), que en este caso corresponde a algún elemento del conjunto T y tiene el valor 51 y tenemos 3 terminales, entonces, 51 mod 3 es 0 correspondiente a la variable x.

- El ultimo gen para generar la primera instrucción, proveniente de la producción $< T >< OP >< T >$, corresponde al segundo operando, en este caso nuevamente es <T>. El valor del gen es 174 y 174 módulo 3 es 0 correspondiente de nuevo a x.

entonces, la primera instrucción es $u_1 = x * x$

2. Repetir el proceso para los genes restantes

   (a) El siguiente gen corresponde al símbolo de inicio, $< S >$ (como en el primer paso) y tiene el valor 229, módulo 4 es 1 correspondiente a $< I >< op >< T >$. La segunda instrucción u_2 estará construida por una instrucción (operando de la izquierda) y una terminal (operando de la derecha). Los pasos para generar u_2 es similar a la anterior, pero, se deben tener algunas consideraciones que se explican a continuación:

   - El gen que corresponde a la producción de operaciones (<op>) tiene el valor 189 y 189 módulo 4 es 1 correspondiente a la operación menos $(-)$
   - El siguiente gen es el primer operando que en este caso es una instrucción (<I>) tiene el valor 8, módulo 1 es 0 (1, porque solo se tiene una instrucción, u_1) , tomando la instrucción u_1 como el primer operando
   - El segundo operando (correspondiente a un terminal) tiene el valor del gen 9, módulo 3 es 0, que es X

   entonces, la segunda instrucción es $u_2 = u_1 - x$.

   (b) El proceso se repite para la tercera instrucción y esta es $u_3 = u_2 + 1$.

Entonces, el SLP generado es:

$$\Gamma = \begin{cases} u_1 & := x * x \\ u_2 & := u_1 - x \\ u_3 & := u_2 + 1 \end{cases} \tag{4.2}$$

Expresión	Valor del gen	Número de Producciones	Índice de la producción seleccionada	No. de instrucción (I)	Producción Seleccionada
<S>	16	4	16 mod 4 = 0	0	<T><op><T>
<op>	18	4	18 mod 4 = 2	0	*
<T>	51	3	51 mod 3 = 0	0	X
<T>	174	3	174 mod 3 = 0	0	X
Instrucción generada $u_1 = \mathrm{x} * \mathrm{x}$					
<S>	229	4	229 mod 4 = 0	1	<I><op><T>
<op>	189	4	189 mod 4 = 1	1	-
<I>	8	1	8 mod 1 = 0	1	u_1
<T>	9	3	9 mod 3 = 1	1	X
Instrucción generada $u_2 = \mathrm{u}_1 - \mathrm{X}$					
<S>	109	4	109 mod 4 = 1	2	<I><op><T>
<op>	200	4	200 mod 4 = 0	2	+
<I>	3	2	3 mod 2 = 1	2	u_2
<T>	134	3	134 mod 3 = 2	2	1
Instrucción generada $u_3 = u_2 + 1$					

Fig. 4.3 Proceso de decodificación de la parte topológica de 4.2

4.2 Operadores Genéticos

Los principales operadores en GP son los operadores de cruce y mutación. Estas secciones explican en detalle cómo se aplican estos operadores en nuestro algoritmo (GSPLGP).

4.2.1 Operador de cruce

Este operador se aplica con probabilidad P_c en dos partes. La primera parte: consiste en intercambiar cada gen con un cruce uniforme, con una probabilidad de 0,8. Después de este procedimiento tenemos cuatro individuos, luego se realiza un torneo en el que se selecciona un punto de muestra aleatorio entre $z = (x_i, y_i) \in \mathbb{R}^n \times Y, 1 \leq i \leq m$. Cada SLP se clasifica por su valor fitness y seleccionamos los dos primeros del ranking. En el caso que tres de los individuos tienen el mismo valor fitness, la selección consiste en tomar uno de ellos y el seleccionado en el segundo lugar es el peor de los cuatro individuos. Los criterios anteriores se utilizan para prevenir la convergencia prematura y mantener la diversidad de las poblaciones. Este primer cruce se hace para aprovechar las características lineales de la evolución gramatical. La segunda parte: consiste en aplicar el operador de cruce para SLP tomando *partes de código efectivas* de cada una e intercambiándolas como se describe en la definición 7. Luego volvemos a aplicar el torneo exactamente igual que antes.

4.2.2 Operador de Mutación

El algoritmo propuesto utiliza tres tipos de operadores de mutación. El proceso de mutación comienza con una probabilidad P_m. En el primer escenario: para cada gen en la parte de la topología, con probabilidad P_{tm} se elige aleatoriamente un nuevo valor en el rango [0,255]. Segunda etapa: se agregan nuevas instrucciones, debido a que una instrucción se genera mediante cuatro genes, necesitamos agregar cuatro genes consecutivamente con probabilidad P_{add} en una posición aleatoria según el mapeo desde el número de la instrucción a la posición de la gramática, mediante $pos = 4 * k$, donde $1 \leq k \leq l$ y la tercera parte: consiste en eliminar una instrucción, esto es, eliminar cuatro genes consecutivamente con probabilidad $P_{eliminar}$ en una posición aleatoria según el mapeo anterior tomando en cuenta que k ahora está en $1 \leq k \leq l-1$.

4.3 Método de penalización

El método de penalización ayuda a GSLPGP a seleccionar buenos candidatos en términos de longitud (número de operaciones) y MSE como padres. Las siguientes subsecciones explican en detalle los mecanismos de selección aplicados y el método de penalización propuesto.

4.3.1 Mecanismo de selección

Para seleccionar μ individuos de la población actual (con tamaño μ) para el proceso de reproducción, se adoptó el mecanismo basado en rango [3]. Este mecanismo permite que los individuos con bajo valor fitness también puedan reproducirse (con una baja probabilidad). Todos los individuos se clasifican, primero, de acuerdo a su valor fitness. Si el individuo más apto se le asigna el rango μ y la peor clasificación es 1, la probabilidad de seleccionar al individuo con el rango j se recalcula linealmente de la siguiente manera:

$$\text{Pr(j)} = \frac{(2-s)}{\mu} + \frac{2(j-1)(s-1)}{\mu(\mu-1)} \tag{4.3}$$

donde s es un parámetro entre 1 y 2 que determina la precisión de la selección. Además, se adopta elitismo para evitar perder a los mejores individuos encontrados durante las generaciones.

4.3.2 Evaluación fitness

Una medida apropiada para evaluar la idoneidad de un SLP es el error cuadrático medio (MSE) (consulte la ecuación 2.4). Básicamente consiste en : dado $z = (x_i, y_i) \in \mathbb{R}^n \times \mathbb{R}, 1 \leq i \leq m$, para cualquier SLP Γ sobre F y T el MSE de Γ es:

$$\mathrm{MSE}_z(\Gamma) = \varepsilon_z(\Phi_\Gamma) = \frac{1}{m}\sum_{i=1}^{m}(\Phi_\Gamma(x_i) - y_i)^2 \tag{4.4}$$

Φ_Γ[1] es la función semántica. Sin embargo, el objetivo de este algoritmo es proporcionar un SLP no solo con un MSE bajo en z, sino también con el número más pequeño de operaciones posibles que se pueda hacer para resolver el problema (es decir, la longitud). Para lograr este objetivo, se aplica una penalización adaptativa a aquellos SLP de mayor longitud. Por lo tanto, la función fitness para un individuo se define de la siguiente manera:

$$F_z(i) = MSE(i) + p_{co}h(i) \tag{4.5}$$

donde $h(i)$ es el número de instrucciones generadas en el proceso topológico, es decir, la longitud del SLP y p_{co} es un coeficiente de penalización. Este coeficiente, p_{co}, se determina al explotar la información de la población utilizando un método adaptativo, lo que evita que el algoritmo produzca innecesariamente SLP complejos y grandes.

Sean $\overline{MSE}$, $\overline{h}$ el MSE y longitud promedio sobre la población actual respectivamente. El coeficiente de penalización es entonces, recalculado en cada generación de la siguiente forma:

$$p_{co} = (\overline{h})/\overline{MSE} \tag{4.6}$$

Donde $q(\overline{h})$ es una función creciente en $\overline{h}$. Tenga en cuenta que si el algoritmo alcanza SLP más altos y más complejos, la definición anterior permite una mayor penalización. Además, con el fin de hacer que la contribución de la penalización no sea mayor al MSE, mediante experimentos preliminares, la función creciente $\overline{h}$ se define como:

$$q(\overline{h}) = \frac{\overline{h}}{h_{\max}^{1.4}} \tag{4.7}$$

[1] La función semántica se describió en la sección 2

donde $h_{\max}$ es la máxima longitud establecida para nuestros SLPs. Tomando las ecuaciones 4.6 y 4.7, el error declarado en la ecuación 4.5 puede reescribirse como:

$$F_z(i) = MSE(i) + \frac{\overline{h}}{h_{\max}^{1.4}\overline{MSE}} * h(i) \tag{4.8}$$

Capítulo 5

Experimentación

El algoritmo propuesto en este trabajo se comparó con el algoritmo propuesto por [4]. En la siguiente sección se describen las funciones objetivo que se usaron y la configuración de ambos algoritmos.

5.1 Funciones objetivo

GSLPGP se ejecutó sobre 2 conjuntos de funciones objetivo. El primer conjunto incluye 3 funciones (ver 5.1) estas fueron usadas por [2].

$$\begin{aligned} F(x,y,z) &= (x+y+z)^2+1 \\ G(x,y,z) &= \frac{1}{2}x+\frac{1}{3}y+\frac{2}{3}z \\ H(x,y,z,w) &= \frac{1}{2}x+\frac{1}{4}y+\frac{1}{6}z+\frac{1}{8}w \end{aligned} \tag{5.1}$$

El segundo conjunto de funciones objetivo está compuesto por una variedad que incluyen funciones trigonométricas, polinómicas y la función mínimo, cada una definida en un dominio determinado. Estas funciones son las siguientes:

$$
\begin{aligned}
f_1(x) &= x^4 + x^3 + x^2 + x \\
f_2(x) &= 2.178x^2 + 3.1416x \\
f_3(x) &= \cos(2x) \\
f_4(x) &= min\left\{\frac{2}{x}, \sin(x) + 1\right\}
\end{aligned}
\tag{5.2}
$$

5.1.1 Detalles de los experimentos.

Las configuraciones y las diferencias de ambos algoritmos se describen en esta sección. Ambos algoritmos utilizaron las siguientes consideraciones:

1. La función // indica la división protegida, esto es x//y regresa x/y si $y \neq 0$ y 1 en cualquier otro caso.

2. Para el primer conjunto de funciones, la muestra de puntos es seleccionada de manera aleatoria en el rango de $[-100, 100]$. Las constantes $c_i, 1 \leq i \leq 6$, toman valores aleatorios en [0,1].

3. Para cada función objetivo, las constantes se fijan antes del comienzo de la primera ejecución y conservan los valores asignados a lo largo de las 100 ejecuciones.

4. Una ejecución consiste en generar un tamaño de población con 500 individuos y ejecutar el algoritmo a través de 500 generaciones.

5. Ambos algoritmos tienen una configuración adecuada que esta descrita en las tablas (5.3, 5.4) con longitud máxima para los SLPs de $h_{\max} = 12$.

6. Una ejecución se considera exitosa si se ha evolucionado un individuo con un valor de MSE inferior a 30.

7. Para el segundo conjunto de funciones objetivos (ver 5.2) las configuraciones son básicamente las mismas descritas anteriormente (ver tabla 5.1) con las siguientes diferencias:

   - El conjunto de constantes es $\{0, 1, 2\}$ para todas las funciones.
   - El conjunto básico de Funciones $F = \{+, -, *, //\}$ es incrementado con otras funciones específicas.

- Algunas funciones tienen un intervalo particular para tomar la muestra de puntos.
- Una ejecución se considera exitosa si se ha evolucionado un individuo con un valor de MSE inferior a 1.

Número de puntos de muestra	30
Conjunto de funciones F	{+,-,*,//}
Variables V	{x,y,z,w}
Constantes C	$\{c_1, ..., c_6\}$
Ejecuciones por función	100

Table 5.1 Configuraciones para el primer conjunto de funciones objetivo

Función	Rango	Conjunto de Funciones
f_1	$[-5,5]$	$F \bigcup \{sqrt\}$
f_2	$[-\pi,\pi]$	$F \bigcup \{\sin,\cos\}$
f_3	$[-\pi,\pi]$	$F \bigcup \{sqrt,\sin\}$
f_4	$[0,15]$	$F \bigcup \{\sin,\cos\}$

Table 5.2 Rango de intervalos para puntos de muestra y funciones establecidas para el segundo grupo de funciones objetivo

El esfuerzo computacional (CE) se define como el número total de operaciones básicas que se han computado hasta el momento. El hecho de establecer cada ejecución en 500 generaciones en lugar del número de operaciones se debe a que, en esencia, ambos algoritmos realizan el mismo número de operaciones excepto por alguna constante. Por lo tanto, se considera un buen esfuerzo computacional para todos los cálculos realizados antes de 500 generaciones.

Para comparar el rendimiento de las estrategias evolutivas se usó el *promedio del mejor fitness* o Mean Best Fitness (MBF), que es el promedio de todas las ejecuciones del mejor valor fitness al final de cada generación. La *tasa de éxito* o Succes Rate (SR) es considerada en problemas con soluciones conocidas para indicar la eficacia algorítmica (consulte [1, 11]) y se determina como la relación entre un número de ejecuciones exitosas con respecto al número total de ejecuciones que se terminan después de un número específico de CE (esfuerzo computacional) en este caso 500 generaciones.

Las tablas 5.3 y 5.4 muestra los parámetros utilizados por los algoritmos y la tabla 5.5 muestra la descripción del equipo donde se realizaron los experimentos.

Población tamaño μ	no. de generaciones	Prob. de cruce (p_c)	Prob. de mutación (p_m)
500	500	0.9	0.15
Primer etapa de mutación(p_{tm})	Segunda etapa de mutación($p_{\text{add/remove}}$)	s	Porcentaje de elitismo p_{elit}
0.6	$p_m/3$	2	2%

Table 5.3 Parámetros del algoritmo propuesto.

Población Tamaño μ	no. de generaciones	Prob. de Cruce(p_c)	Prob. de mutación (p_m)	Radio de reproducción
500	500	0.9	0.05	0.05

Table 5.4 Parámetros del algoritmo [4] (SLPGP)

Procesador	Core i5 7th gen
Ram	8 Gb
SO	Ubuntu
Lenguaje	c++ no sin opciones de optimización

Table 5.5 Descripción del ambiente

Capítulo 6

Resultados Experimentales

Esta sección presenta los resultados obtenidos a través de las experimentaciones realizadas (descritas en la sección anterior).

6.1 Resultados para el primer conjunto de funciones objetivo

En las Figuras 6.1, 6.2 y 6.3, se muestra la gráfica del mejor fitness medio (MBF) contra las generaciones sobre las funciones objetivo F, G y K, para las dos algoritmos considerado (SLPGP y GSLPGP). La tabla 6.1 muestra la tasa de éxito de 100 ejecuciones independientes, además de la altura promedio que consigue nuestro algoritmo contra la longitud de 12 que se mantiene fija para SLPGP. Teniendo en cuenta las ejecuciones exitosas y el uso de GSLPGP, mostramos en la tabla 6.2 el mejor fitness medio y el mejor fitness absoluto medio (ABF) obtenida, después de la máxima generación establecida, de 500.

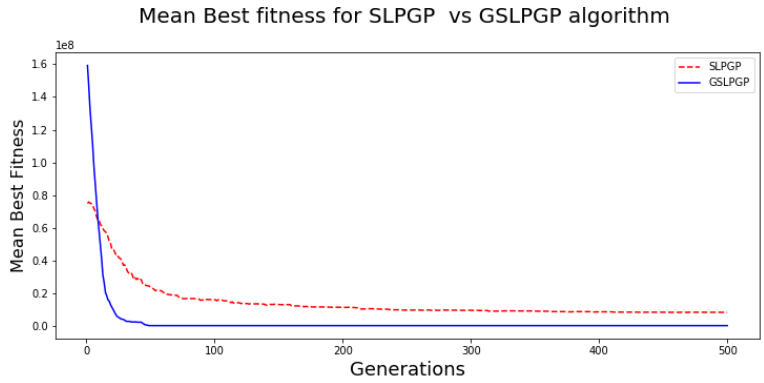

Fig. 6.1 Mejor fitness medio contra las generaciones para $F = (x+y+z)^2+1$

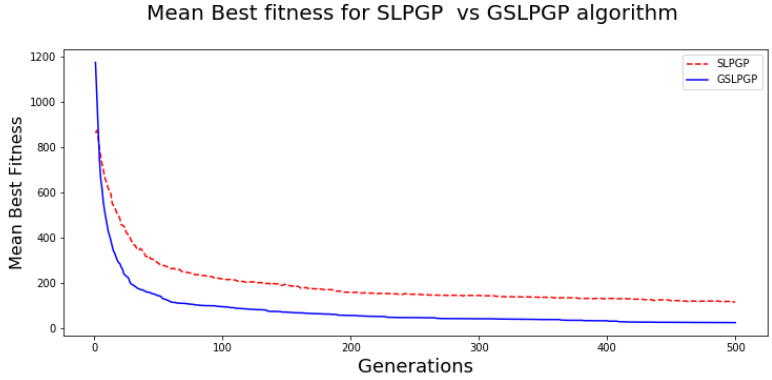

Fig. 6.2 Mejor fitness medio contra las generaciones para $G = \frac{1}{2}x+\frac{1}{3}y+\frac{2}{3}z$

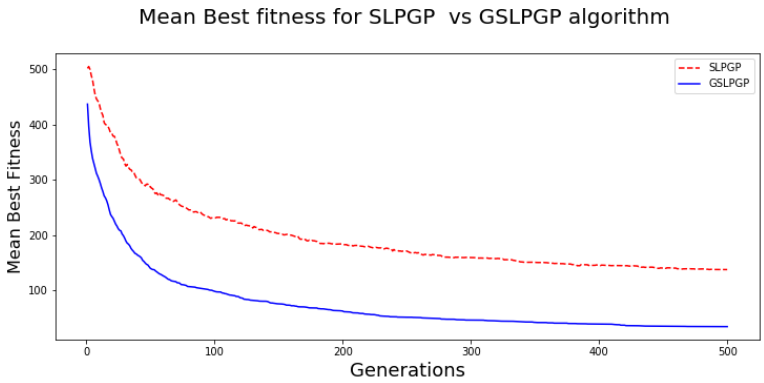

Fig. 6.3 Mejor fitness medio contra las generaciones para $K = \frac{1}{2}x+\frac{1}{4}y+\frac{1}{6}z+\frac{1}{8}w$

Función	SLPGP	GSLPGP	Longitud promedio
F	52	98	9.1
K	21	25	10.16
H	4	32	9.68

Table 6.1 Tasa de éxito calculada en 100 ejecuciones independientes para SLPGP y GSLPGP

	SLPGP		GSLPGP	
función	MBF	ABF	MBF	ABF
F	1.46e+00	1.21e+00	3.68e-01	6.07e-01
K	1.83e+01	4.28e+00	7.21e+00	2.68e+00
H	2.00e+01	4.47e+00	1.80e+01	4.25e+00

Table 6.2 Mejor fitness medio y mejor fitness absoluto medio calculada sobre las ejecuciones exitosas para SLPGP y GSLPGP

Como conclusión, para las funciones objetivo estudiadas, el uso de nuestro algoritmo GSLPGP es más efectivo que el algoritmo SLPGP.

6.2 Resultados para el segundo conjunto de funciones objetivo

Para este segundo grupo de funciones objetivo, en los resultados no se presentan gráficas, ya que fundamentalmente son similares, sin embargo, los resultados se simplifican en tablas que nos dan una mejor comparación de los algoritmos. La tabla 6.3 muestra la tasa de éxito de 100 ejecuciones independientes, además de la altura promedio que consigue nuestro algoritmo contra la longitud de 12 que se mantiene fija para SLPGP. Considerando las ejecuciones exitosas y usando nuestro algoritmo, se muestra en la tabla 6.4 el mejor fitness medio y el mejor fitness absoluto obtenido (ABF) después de la máxima generación establecida, de 500.

Función	SLPGP	GSLPGP	longitud
f_1	64	91	7.13
f_2	21	46	7.38
f_3	100	83	8.03
f_4	100	100	5.55

Table 6.3 Tasa de éxito calculada en 100 ejecuciones independientes para SLPGP y GSLPGP

	SLPGP		GSLPGP	
Función	MBF	ABF	MBF	ABF
f_1	6.14e-28	2.48e-14	1.83e-27	4.28e-14
f_2	6.57e-01	8.11e-01	7.55e-01	8.69e-01
f_3	2.00e+01	4.47e+00	1.80e+01	4.25e+00
f_4	1.21e-01	3.48e-01	5.80e-01	7.62e-01
f_5	3.82e-02	1.96e-01	2.65e-02	1.63e-01

Table 6.4 Mejor fitness promedio y mejor fitness absoluto promedio calculado sobre las ejecuciones exitosas para SLPGP y GSLPGP

Como conclusión: para las funciones objetivo estudiadas, el uso del algoritmo GSLPGP es más efectivo en la mayoría de los casos.

Capítulo 7

Conclusiones

Hemos experimentado con Straight Line Programs, programación genética y evolución gramatical en un enfoque combinado. Este estudio propone una nueva combinación de GP y Straight Line Programs que necesita un esfuerzo mínimo para personalizar el tamaño de los SLP. El algoritmo propuesto (GSLPGP) utiliza codificación de la gramática para la representación de la topología en un cromosoma. Para simplificar los SLP a través del proceso de evolución, el sistema utiliza un novedoso enfoque de penalización adaptativa que fomenta topologías más pequeñas que conducen a una mayor capacidad de generalización cuando es necesario. Para evaluar el rendimiento del algoritmo propuesto, se realizaron experimentos extensivos en dos conjuntos diferentes de funciones objetivo y los resultados se comparan con otros algoritmos que usan SLP en GP (SLPGP) conocidos y de vanguardia en la literatura. Los resultados computacionales demuestran la superioridad de nuestro algoritmo sobre el otro método, ya que proporciona el mejor rendimiento en términos de MSE y operaciones necesarias para resolver el problema de regresión simbólica.

Como trabajos futuros en nuestra investigación, está de manera natural la combinación del enfoque GP desarrollado aquí con otros métodos tales como optimización por gradiente descendiente y la cooperación de ambos. Además, de realizar más experimentación sobre otras funciones objetivos. Por último, realizar la evolución de las constantes, ya que se podría presentar mejoras en el sistema con este enfoque.

Referencias

[1] M. Jelasity A. Eiben. “A critical note on experimental research methodology in EC”. In: *In Proc. of the Congress on Evolutionary Computation* (2002), pp. 582–587.

[2] W.F. Punch A. Topchy. “Faster genetic programming based on local gradient search of numeric leaf values”. In: *In Proc. of Genetic and Evolutionary Computation Conference* (2001), pp. 155–162.

[3] J.E. Smith A.E. Eiben. *Introduction to Evolutionary Computing*. Springer, Berlin, Heidelberg, 2003.

[4] J. L. Montaña C. L. Alonso J. Puente. “Straight Line Programs: A new Linear Genetic Programming Approach”. In: *2008 20th IEEE International Conference on Tools with Artificial Intelligence* 1.2 (2008), pp. 519–520. URL: https://ieeexplore.ieee.org/document/4669818.

[5] J. R. Coza. *Genetic Programming II*. MIT Press,MA, 1994.

[6] J. R. Coza. *Genetic Programming: on the Programming of Computer by Means of Natural Selection*. MIT Press,MA, 1992.

[7] J.J. Freeman. “Genetic Programming 1998: Proceedings of the Third Annual Conferenc”. In: *IEEE* (1998), pp. 72–77.

[8] J.E. Morgentern M. Giusti J. Heintz and L.M. Pardo. “Straight Line Programs in Geometric Elimination Theory”. In: *Journal of Pure and Applied Algebra* 124 (1997), pp. 121–146.

[9] C. Ryan M. O'Neill. *GRAMMATICAL EVOLUTION: Evolutionary Automatic Programming in an Arbitrary Language*. Springer, 2003.

[10] K. Mulmuley. "A parallel algorithm to compute the rank of a matrix over an arbitrary field". In: *Combinatrica pp.* 7 (1987), pp. 101–104.

[11] T.Back. *Evolutionary Algorithms in Theory and Practice*. Oxford: Oxford University Press, 1996.

[12] R. E. Keller W. Banzhaf P. Nordin and F. D. Francone. *Genetic Programming An Introduction: On the Automatic Evolution of Computer Programs and its Applications*. Morgan Kaufmann, 1998.

[13] R. E. Keller W. Banzhaf P. Nordin and F. D. Francone. *Genetic Programming - An introduction:On the Automatic Evolution of Computer Programs and its Applications*. Morgan Kaufmann, San Francisco,California, 1998.

[14] M.L Wong and K.S Leung. "Inductive logic programming using genetic algorithms". In: *Advances in Artificial Intelligence - Theory and Application II* 2 (1995), pp. 119–124.