UNIVERSIDAD MICHOACANA DE SAN NICOLÁS DE HIDALGO

DIVISIÓN DE ESTUDIOS DE POSGRADO DE LA FACULTAD DE INGENIERÍA ELÉCTRICA

UNA ESTRATEGIA PARA PROGRAMACIÓN GENÉTICA BASADA EN SUBMUESTREOS INCREMENTALES

TESIS

QUE PARA OBTENER EL GRADO DE:

MAESTRO EN CIENCIAS EN INGENIERÍA ELÉCTRICA

PRESENTA:

MARCO ANTONIO PACHECO ALVAREZ

DR. EN CIENCIAS COMPUTACIONALES

MARIO GRAFF GUERRERO

DIRECTOR DE TESIS

DOCTOR EN INGENIERÍA ELECTRÓNICA Y ELÉCTRICA

JAIME CERDA JACOBO

CO-DIRECTOR DE TESIS

MORELIA, MICHOACÁN, JULIO 2014

DIVISIÓN POSGRADO

DIVISIÓN DE ESTUDIOS DE POSGRADO DE LA FACULTAD DE INGENIERÍA ELÉCTRICA

Resumen

La identificación de sistemas es el proceso de encontrar un modelo matemático en base a las mediciones de la relación entrada/salida del sistema. Programación Genética (PG) es una técnica evolutiva utilizada para evolucionar expresiones matemáticas. GP es una técnica de aprendizaje supervisado, es decir, las expresiones evolucionadas se basan en las mediciones realizadas al sistema en estudio. En consecuencia, GP es una metaheurística apropiada para resolver la tarea de identificación de sistemas.

Una estrategia tradicional en GP para la identificación de sistemas es tomar todas las medidas (observaciones) durante la evolución del modelo. Contrario a esto, en esta tesis, se propone una nueva metodología que se basa en utilizar un subconjunto parcial de las observaciones, e incrementar este subconjunto hasta llegar al conjunto total de observaciones. El sistema de programación genética propuesto con esta tesis implementa esta estrategia; este sistema se llama Incremental Genetic Programming (IGP).

IGP se compara con un sistema PG tradicional y un software comercial para la identificación de sistemas llamado Eureqa. IGP supera al sistema PG tradicional y Eureqa; obteniendo mejores modelos, así como una mejor convergencia, y requiere menos recursos de cálculo para derivar el modelo. Sin embargo, los modelos obtenidos por IGP tienen una mayor complejidad que los obtenidos por Eureqa. Como trabajo futuro, IGP se puede mejorar con técnicas que permitan la evolución de modelos más simples.

Palabras clave: *identificación de sistemas, Programación Genética, modelo, IGP, Eureqa*

Abstract

System identification is the process of finding a mathematical model based on the measurements of the system's inputs/outputs relationship. Genetic Programming (GP) is an evolutionary technique used to evolve mathematical expressions. GP is a supervised machine learning technique, that is, the expressions evolved are based on the measurements made to the system under study. Consequently, GP is a appropriate heuristic to solve the task of system identification.

A traditional strategy in GP for system identification is to take all the measurements (observations) during the evolution of the model. Contrary to this, in this thesis, a new methodology, based on using a subset of the observations, wich grows incrementally until reaching the total set of observations, is proposed in this thesis. The Genetic Programming System that implements this strategy is called Incremental Genetic Programming (IGP).

IGP is compared against a traditional GP system and a commercial system identification software called Eureqa. IGP outperforms the traditional GP system and Eureqa; it obtains better models as well as convergence, and requires less computational resources to derive the model. Nonetheless, the models obtained by IGP have higher complexity than the ones obtained by Eureqa. As future work, IGP can be enhanced with techniques that allow the evolution of simpler models.

Key words: *system identification, Genetic Programming, model, IGP, Eureqa*

Contenido

Lista de Figuras

Lista de Tablas

Lista de Algoritmos

Lista de Símbolos

α	Factor de escalamiento con valor de 1,000,000.
β	Incremento.
ω	Número de veces para submuestrear las observaciones.
μ	Media aritmética.
aptitud	Unidad de medida para comparar resultados.
costo	Unidad de medida para determinar condición de paro.
$Costo\_Max$	Costo máximo.
F	Conjunto de Funciones.
$f(x)$	Comportamiento del modelo encontrado.
n	Número de nodos que tiene un individuo.
N	Número de observaciones del sistema a modelar.
O	Conjunto de observaciones.
PG	Programación Genética.
t	Tamaño del individuo.
T	Conjunto de Terminales.
Tam_Max	Tamaño máximo que puede tener un individuo.
$U(0,1)$	Número aleatorio con valor entre 0 y 1.
y	Observaciones del sistema a modelar.

Lista de Publicaciones

- Publicados

  Marco A. Pacheco, Mario Graff and Jaime Cerda. "A Fitness Case Strategy in Genetic Programming to Improve System Identification" XV Reunión de Otoño de Potencia, Electrónica y Computación 2013. Morelia, México. Noviembre 2013.

Capítulo 1

Introducción

Esta tesis plantea la aplicación de Programación Genética (PG) para la identificación de sistemas [Flores and Graff, 2005], desarrollando un sistema capaz de modelar un sistema físico en base a sus observaciones, es decir, a su comportamiento. Un sistema es una combinación de componentes que actúan juntos teniendo un comportamiento que lo define y por lo tanto una estructura que hace posible dicho comportamiento.Cuando se conoce esa estructura se dice que se ha encontrado el modelo del sistema.

Una vez que se tiene un modelo se pueden realizar acciones de control o simulación. Por ejemplo, el modelo de un transformador se puede usar como huella digital del mismo, obteniendo su comportamiento en un estado normal, es decir, cuando no presente ningún tipo de fallos, con el objetivo de identificar posibles anomalías después de un determinado tiempo de operación o ante el sometimiento a condiciones de operación estresantes para el equipo, que pueden originar fallos internos en el mismo.

Normalmente obtener el modelo de un sistema es un proceso al que se le dedica mucho tiempo y esfuerzo, debido a que es un proceso complejo, ya que el modelo encontrado debe ser preciso, asegurando un comportamiento similar al del sistema.

Existen diferentes técnicas para la identificación de sistemas, por ejemplo *Mínimos Cuadrados*. El problema con este tipo de técnicas es la necesidad de conocer parcial o totalmente la estructura del sistema, por lo que para esta tesis se ha omitido el estudio y experimentación con estas técnicas.

Es interesante la utilización de PG, ya que al ser una técnica de computación evolutiva y al igual que otras técnicas de este tipo, su proceso de evolución es automático, es decir, la búsqueda del modelo se realiza sin la necesidad de la intervención del usuario, sólo es necesario proporcionar el conjunto de observaciones del sistema que se pretende modelar. Además, no es necesario tener conocimiento alguno del sistema que se pretende modelar.

PG trabaja con expresiones aritméticas, que representan los modelos generados en el proceso de la evolución. La forma en que PG evoluciona dichas expresiones es a través del aprendizaje supervisado, esto significa que aprende en base a un comportamiento esperado, por lo que son necesarias las observaciones del sistema.

En este trabajo se propone la estrategia de utilizar submuestreos incrementales, esto es, buscar modelos para subconjuntos de observaciones, los cuales van incrementando su tamaño conforme se van encontrando modelos aceptables para dichos subconjuntos, con el fin de mejorar el rendimiento de PG.

Para probar la efectividad de esta propuesta, se han encontrado modelos para nueve señales de transformadores de potencia, utilizando las respuestas generadas por la prueba de Sweep Frequency Response Analysis (SFRA) [Avalos, 2008] utilizada para la identificación de fallas incipientes en transformadores de potencia trifásicos que involucren cambios en la geometría interna de sus devanados y/o núcleo. La prueba de SFRA genera para cada transformador 9 señales de respuesta que reflejan la condición interna de 6 devanados y su núcleo.

Esta no es la primera vez que se utiliza PG para encontrar modelos, por ejemplo en [Graff et al., 2013] se hace uso de PG para hacer predicciones en series de tiempo, en [Flores and Graff, 2006] y [Flores and Graff, 2005] se buscan modelos para diferentes sistemas, en [Schmidt M., 2009] se realizan experimentos para la identificación de sistemas físicos, desarrollando una herramienta de software llamado *Eureqa*, el cual se utilizó en esta tesis para la comparación de sus resultados con los obtenidos con el sistema desarrollado.

Una estrategia para mejorar búsqueda de modelos es evaluar menos casos *aptitud* como en [Giacobini et al., 2002], buscando agilizar el proceso de evolución. Por lo que la estrategia propuesta en este trabajo se basa en esta idea, agilizar el proceso de

evolución prestando atención en los casos *aptitud*.

1.1. Objetivo

El objetivo general de esta tesis es desarrollar un sistema que utilice PG para encontrar modelos de sistemas físicos a partir de sus observaciones.

Los objetivos particulares son los siguientes:

- Mejorar el rendimiento de PG utilizando una evaluación basada en submuestreos incrementales.
- Identificar sistemas cuando se tienen cientos de observaciones.
- Identificar sistemas de fenómenos físicos que presenten altas no linealidades.

1.2. Descripción de la Tesis

En este capítulo se mencionó el objetivo que se pretende alcanzar con el desarrollo del presente trabajo, así como una introducción a la identificación de sistemas y el trabajo relacionado.

En el capítulo 2 se describe la identificación de sistemas mediante Programación Genética, describiendo de manera general y mediante algoritmos, los procedimientos necesarios para la obtención de un modelo, así como la unidad de medida utilizada para compararlos.

En el capítulo 3 se analiza el sistema desarrollado utilizando la estrategia propuesta en esta tesis para la identificación de sistemas (IGP), terminando con la descripción de un software llamado *Eureqa*, que fue usado para comparar sus resultados obtenidos contra los resultados obtenidos por IGP.

En el capítulo 4 se presentan los valores para los datos iniciales utilizados por IGP, las señales con las que se llevaron a cabo las pruebas, los resultados gráficos

obtenidos tanto del comportamiento de los modelos encontrados como algunos árboles que representan las funciones de dichos comportamientos.

En el capítulo 5 se presentan las conclusiones obtenidas por la presente tesis, así como su aportación y el trabajo futuro para mejorar los resultados.

Capítulo 2

Identificación de sistemas mediante Programación Genética

La identificación de sistemas es el proceso de encontrar un modelo matemático a partir de las observaciones del sistema a identificar. Para definir el modelo de un sistema, primeramente se debe tener acceso a sus observaciones.

Programación Genética es una técnica de computación evolutiva que se basa en la teoría de la evolución de Darwin [Koza, 1992], es decir, es una heurística basada en la evolución de las especies.

De manera general, un algoritmo de PG sigue el siguiente esquema:

- Generar un conjunto de constantes.
- Establecer un conjunto de variables
- Establecer un conjunto de funciones
- Generar una población inicial.
- Evaluar población.
- Mientras no se cumpla el criterio de terminación:
  - Generar nuevo individuo mediante operadores genéticos.

  - Insertar nuevo individuo en la población.

- Mostrar el modelo obtenido.

Los individuos son expresiones matemáticas representados por un árbol de expresiones. Dichas expresiones están compuestas por *Funciones* y *Terminales*. El conjunto de *Funciones* puede estar conformado por funciones aritméticas, trigonométricas, entre otras, por ejemplo:

- Funciones aritméticas: suma, resta, multiplicación, división

- Funciones trigonométricas: seno, coseno, tangente.

El conjunto de *Terminales* está compuesto por constantes y/o variables, donde las variables son las entradas que recibe el sistema a modelar. A este conjunto también se le conoce como *Funciones* de aridad 0.

En el Algoritmo 1 se muestra el funcionamiento general de un programa que hace uso de PG. Este algoritmo recibe 3 parámetros, el número de generaciones que se realizará la evolución, el tamaño de la población y las observaciones del sistema a modelar. Recordando lo que se mencionó anteriormente, primero es necesario generar aleatoriamente un conjunto de constantes, establecer el conjunto de variables y el conjunto de funciones, las cuales se inicializan en las líneas 1 - 3. Enseguida se procede con la creación de la población, como se muestra en la línea 4, agregando como parámetro el tamaño de la población, esta población evolucionará para producir nuevos individuos para encontrar un modelo similar al buscado. Después, en la línea 6 se indica el número de veces que se realizarán las acciones de las siguientes líneas, determinadas por el parámetro *generaciones*. En una generación se deben de generar tantos individuos como el tamaño de la población, por lo que en la línea 8 se genera un ciclo, con el que se generaran los nuevos individuos que serán creados mediante algún operador genético, como se muestra en la línea 9, para después ser insertado dentro de la población, como indica en la línea 10. Finalmente, una vez que se cumpla el número de *generaciones*, en la línea 13 se retorna el mejor individuo de la población, esto es, el mejor modelo encontrado.

Algoritmo 1 Algoritmo Evolutivo inspirado en TinyGP

```
AlgoritmoEvolutivo(generaciones, tamaño_población, observaciones)
 1  constantes ← Genera_Constantes(rango_inicial, rango_final, cantidad)
 2  variables ← Establece_Variables(lista_variables)
 3  funciones ← Establece_Funciones(lista_funciones)
 4  población ← Crea_Población(tamaño_población)
 5  i ← 1
 6  mientras i ≤ generaciones
 7    j ← 1
 8    mientras j ≤ tamaño_población
 9      individuo ← AplicaOperadorGenético(población)
10      población ← Insertar(población, individuo)
11      j ← j + 1
12    i ← i + 1
13  regresar Mejor(población)
```

2.1. Inicialización

Los primeros pasos para poder hacer uso de un algoritmo de programación genética consisten en generar un conjunto de constantes, establecer un conjunto de variables y funciones para poder crear la población inicial, donde una población esta conformada por un conjunto de individuos. Uno de los parámetros importantes es el tamaño máximo de dichos individuos. Este límite puede definirse de dos maneras:

- Definiendo un número máximo de elementos o nodos.
- Definiendo una profundidad máxima del árbol.

Normalmente se utilizan dos tipos de árboles para la representación de los individuos, los árboles *full* y los árboles *grow*.

- Un árbol *full* tiene la característica de mantener todos sus nodos *Terminales* a la misma profundidad, similar a un árbol balanceado. En la Figura 2.1 se muestra el ejemplo de un árbol *full*, el cual representa la expresión $f(x) = (-(* \ 0.5 \ 3)(+ \ x \ 7.2))$, escrita en notación prefija, siendo esta notación común en [Poli et al., 2008] para la representación de los árboles.
- Un árbol *grow* a diferencia del anterior es que puede generar árboles similares a los balanceados así como no balanceados. En la Figura 2.2 se muestra el ejemplo de un árbol *grow* representando la expresión $f(x) = (- \ (- \ (* \ 0.5 \ 3) \ x) \ 7.2)$ cuya función es la misma a la anterior pero representada con una expresión diferente. Como se puede observar, los nodos *Terminales* de este árbol no están a la misma profundidad, aunque existe la posibilidad de que un árbol *grow* sea igual a un árbol *full*.

En el Algoritmo 2 se muestra la creación de un árbol *full*. Como se puede observar es un método recursivo, que recibe como parámetro la *profundidad* del árbol a generar. En la línea 1 se comprueba si la llamada a este método es de profundidad 0, si se cumple esta condición, el método retornará un nodo hoja, también conocido como *Terminal*, como se indica en la línea 3. En la línea 5 se le asigna a la variable *func*

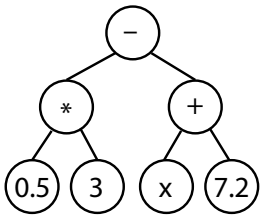


Figura 2.1: Árbol *full*

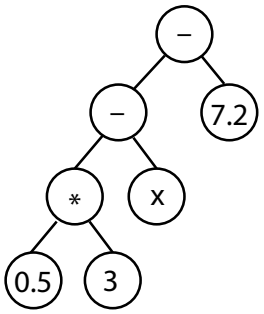


Figura 2.2: Árbol *grow*

una función seleccionada de manera aleatoria del conjunto de *Funciones*. En la línea 6, a la variable *núm_arg* se le asigna el número de argumentos que necesita la función antes seleccionada. En la línea 7 se declara la variable *a_full* como tipo de dato *Lista* y en la línea 8 se esta agregando la función *func* a la lista *a_full*. En la línea 9 se genera un recorrido que va desde i hasta el número de argumentos, y de la línea 10 a 13 se hace una llamada recursiva decrementando la profundidad. Finalmente, en la línea 13 se regresa la variable *a_full* que representa el árbol *full* generado.

En el Algoritmo 3 se muestra la creación de un árbol *grow*, donde se puede observar que la única diferencia con respecto al Algoritmo 2 es que la línea 5 presenta una condición, que si se cumple, se estarían generando ramas completas, pero en el caso contrario se generaría un nodo *Terminal*, lo que provocaría la generación de árboles *grow*.

El Algoritmo 4 muestra las instrucciones para la creación de una población, recibiendo como parámetro el tamaño de la misma. Este algoritmo utiliza el método

Algoritmo 2 Creación de un árbol *full*

```
ÁRBOL_FULL(profundidad)
 1  si profundidad = 0
 2     entonces
 3              regresar HOJA_ALEATORIA
 4     sino
 5              func ← FUNCIÓN_ALEATORIA(nfunc)
 6              núm_arg ← NARGS(func)
 7              a_full ← Lista
 8              a_full ← AGREGA(a_full, func)
 9              para i ∈ núm_arg
10                tmp ← Lista
11                tmp ← ÁRBOL_FULL(profundidad − 1)
12                a_full ← AGREGA(a_full, tmp)
13              regresar a_full
```

Algoritmo 3 Creación de un árbol *grow*

```
ÁRBOL_GROW(profundidad)
 1  si profundidad = 0
 2     entonces
 3              regresar HOJA_ALEATORIA
 4     sino
 5              si ALEATORIO < 0.5
 6                 entonces
 7                          func ← FUNCIÓN_ALEATORIA(nfunc)
 8                          núm_arg ← NARGS(pos)
 9                          a_grow ← Lista
10                          a_grow ← AGREGA(a_grow, func)
11                          para i ∈ núm_arg
12                            tmp ← Lista
13                            tmp ← ÁRBOL_GROW(profundidad − 1)
14                            a_grow ← AGREGA(a_grow, tmp)
15                          regresar a_grow
16                 sino
17                          regresar HOJA_ALEATORIA
```

conocido como *ramped half-and-half* [Poli et al., 2008] que consiste en generar la población con árboles *full* y *grow*. En la línea 1 se declara a la población como una lista. En las líneas 2-4 se indica que se está utilizando el método *Árbol_Full* para generar la mitad de la población, mientras que en las líneas 5 a 7 se utiliza el método *Árbol_Grow* para generar la otra mitad de la población. Finalmente en la línea 8 se retorna la población generada.

Algoritmo 4 Creación de la población

```
CREA_POBLACIÓN(tamaño_población)
1  población ← Lista
2  para i en númeroIndividuos/2
3    nuevo ← ÁRBOL_FULL(ALEATORIO)
4    población ← INSERTAR(población, nuevo)
5  para j en númeroIndividuos/2
6    nuevo ← ÁRBOL_GROW(ALEATORIO)
7    población ← INSERTAR(población, nuevo)
8  regresar población
```

2.2. Operadores Genéticos

Los operadores genéticos son procedimientos que se les aplican a los individuos que conforman la población, con el objetivo de generar nuevos individuos a partir de los existentes, generando nuevas posibles soluciones para el problema que se pretende resolver. Estos operadores son los encargados de llevar a cabo la evolución en PG. A continuación se describe un algoritmo que sólo hace uso de la cruza y mutación, siendo estos los únicos operadores utilizados para la evolución de los individuos en este algoritmo. Posteriormente se describen de manera gráfica el funcionamiento de dichos operadores.

El Algoritmo 5 tiene como propósito generar un nuevo individuo a partir dos individuos de la población, es decir, mediante individuos llamados padres, ya que con las partes de los padres se genera un nuevo individuo llamado hijo. Como se puede observar, este algoritmo recibe como parámetro la población y sólo hace uso de dos operadores genéticos: *Cruza* y *Mutación*, para los cuales se manejan las probabilidades 90 % y 10 % de ocurrencia respectivamente.

Algoritmo 5 Algoritmo que aplica operadores genéticos

```
AplicaOperadorGenético(población)
1  si U(0,1) < 0.9
2     entonces
3              Padre1 ← Selección_Torneo(población, mejor)
4              Padre2 ← Selección_Torneo(población, mejor)
5              hijo ← Cruza(Padre1, Padre2)
6     sino
7              Padre1 ← Selección_Torneo(población, mejor)
8              hijo ← Mutación(Padre1)
9  regresar hijo
```

En la línea 1 se utiliza la probabilidad de realizar una *Cruza*, donde U(0,1) es un número generado aleatoriamente con una función de distribución uniforme entre 0 y 1. En las líneas 2 a 5 se declara la acción de crear una *Cruza*, para lo que es necesario (líneas 3 y 4) seleccionar 2 padres, para poder utilizar dicho método como se puede observar en la línea 5, donde se genera un hijo con los padres antes seleccionados. Cuando la probabilidad de la primer línea no se cumple, como se indica en las líneas 6 a 8, se genera una *Mutación*, donde sólo es necesario 1 padre, el cuál se obtiene en la línea 8. La técnica utilizada para la selección de los padres es *Selección_Torneo*, la cual se explica más adelante. Para finalizar este método, en la línea 9 se regresa el hijo generado, ya sea por *Cruza* o *Mutación*.

A continuación se describen de manera gráfica la generación de individuos mediante los operadores genéticos más usados.

2.2.1. Cruza

El operador cruza funciona de la siguiente manera. El primer paso es seleccionar dos individuos de la población, denominémosles *Padre1* y *Padre2*. Después aleatoriamente se seccionan un nodo de *Padre1*, al que llamaremos p_1 y un nodo de *Padre2* que llamaremos p_2, después, se duplica a *Padre1*, colocando en la posición del nodo p_1 el contenido del nodo p_2, generando de esta manera un árbol que representa la cruza entre los dos árboles antes mencionados. Por ejemplo en la Figura 2.3 se muestran dos individuos, que por coincidencia son del mismo tamaño, donde en *Padre1* se seleccionó al nodo p_1 con la función *abs* y en *Padre2* se seleccionó al nodo *2* con la función *cos*, enseguida de duplica a *Padre1*, insertando el nodo p_2 con su contenido en la posición del nodo p_1, generando un nuevo árbol llamado *hijo* con partes de ambos padres.

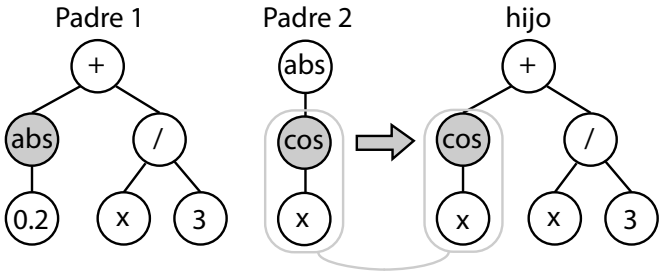


Figura 2.3: Cruza

2.2.2. Mutación Subárbol

Producir un nuevo individuo haciendo uso del operador Mutación subárbol es muy similar al procedimiento anterior, la diferencia está en que el *Padre1* es seleccionado de la población mientras que el *Padre2* se crea de forma aleatoria, en la Figura 2.4 se muestra un ejemplo, donde del *Padre1* se ha seleccionado el nodo sombreado con

valor ***abs***, mientras que del *Padre2* se ha seleccionado el nodo sombreado con valor **-**, produciendo el individuo que corresponde a la Cruza llamado *hijo*, como se muestra en la Figura 2.4.

Uno de los puntos a tomar en cuenta con estos procedimientos es la cantidad de nodos que tiene cada individuo, ya que en el proceso de evolución, esta cantidad puede crecer de manera rápida, por lo que en este trabajo se procesan los nuevos individuos mientras su tamaño no exceda un límite máximo.

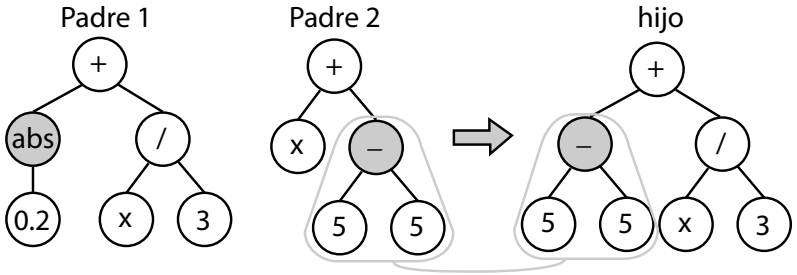


Figura 2.4: Mutación subárbol

2.2.3. Mutación Puntual

El proceso que realiza la Mutación Puntual es diferente a la mutación anterior. Para este caso se selecciona un individuo de la población, para posteriormente recorrerlo nodo a nodo. En dicho recorrido existe una probabilidad de que el nodo en el que se esta revisando pueda ser reemplazado por otro nodo de la misma aridad, por ejemplo, si el nodo contiene una suma(+), sólo puede ser reemplazado por una resta(**-**), multiplicación(*), o división(/), que son operadores algebraicos con el mismo número de operandos. Aunque depende de las funciones que se estén utilizando en el programa genético. En la Figura 2.5 se ejemplifica el recorrido de los nodos, donde se muestran de forma sombreada los nodos donde la probabilidad de ser modificados se ha cumplido y el individuo producido con sus respectivos reemplazos, esto es, el nodo + ha sido reemplazado por el nodo /, mientras que el nodo ***abs*** ha sido reemplazado por el nodo ***tan***.

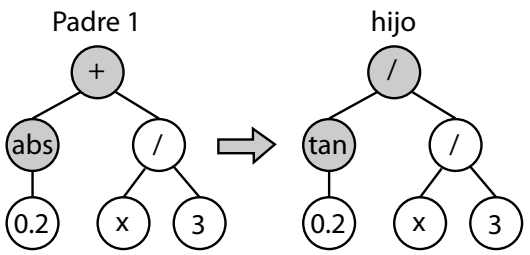


Figura 2.5: Mutación puntual

2.3. Otros Operadores Genéticos

Ahora se presentan otros operadores genéticos, los cuales no son utilizados por el sistema IGP en la evolución de su población, pero se ha decido incluirlos porque pueden ser utilizados para ampliar el número de combinaciones al generar nuevos individuos.

2.3.1. Clonación (Reproducción)

Este es el operador más simple de todos, donde se selecciona un individuo de la población y se duplica, teniendo de esta manera dos individuos iguales dentro de dicha población, como se puede observar el ejemplo de la Figura 2.6.

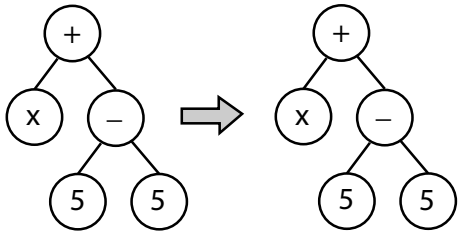


Figura 2.6: Clonación

2.3.2. Permutación

Para el caso de este operador, se selecciona un individuo de la población, escogiendo un nodo al azar, donde las ramas del nodo seleccionado se permutarán, como se muestra en la Figura 2.7. En dicha figura se selecciona el nodo sombreado que contiene la función con el símbolo / que corresponde a la división, por lo que en este caso, sus hojas son permutadas, es decir, el nodo con la variable $\boldsymbol{x}$ toma el lugar del nodo con la constante $\boldsymbol{3}$ y viceversa, produciendo de esta manera un nuevo individuo.

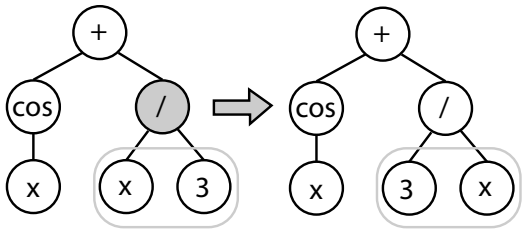
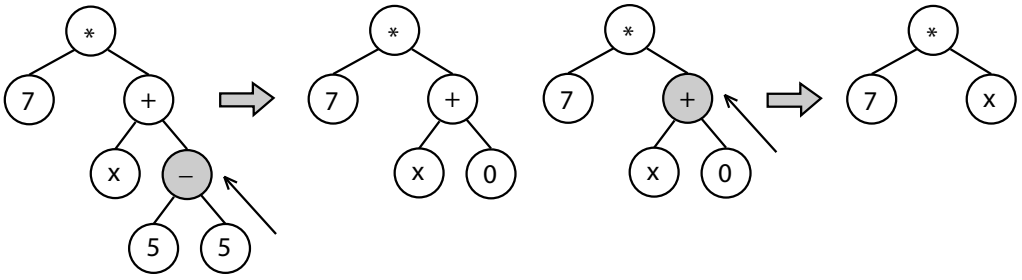


Figura 2.7: Permutación

2.3.3. Simplicación

Este operador cumple con el objetivo de simplificar una árbol, esto es, reducir su tamaño. Aunque esta operación no garantiza una mejor convergencia, si provoca una reducción en la diversidad de la población. A continuación se explica el funcionamiento de este operador. En la Figura 2.8 se muestra la reducción de un individuo, donde en el inciso **a** se muestra el primer paso, la resta del nodo sombreado producirá una nueva constante, modificando de esta manera la estructura del individuo seleccionado, enseguida continúa de manera recursiva con su reducción, ahora seleccionando el nodo sombreado con el símbolo +, que realizará la suma de $\boldsymbol{x}$ con la constante $\boldsymbol{0}$, provocando nuevamente que el individuo sea modificado, arrojando como nuevo individuo un árbol de 3 nodos.

A continuación se muestran figuras donde se ejemplifican los casos en que puede ocurrir una reducción.



(a) Primera reducción. (b) Segunda reducción

Figura 2.8: Reducción de árboles por medio del operador de Edición.

La suma de dos constantes se reduce a un sólo nodo, que es otra constante, como se muestra en la Figura 2.9.

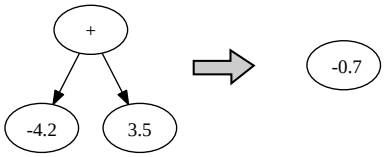


Figura 2.9: Suma de constantes

La resta de dos contantes, da como resultado otra constante, como se puede ver en la Figura 2.10.

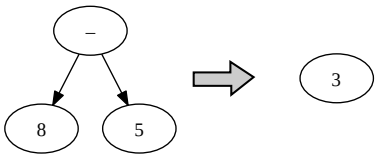


Figura 2.10: Resta de constantes

La resta de dos variables, arroja como resultado otra constantes, la cual es cero, como se observa en la figura 2.11.

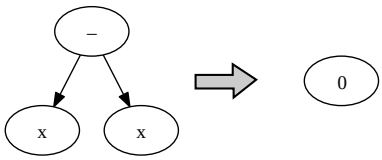


Figura 2.11: Resta de variables

La multiplicación de dos constantes produce otra constante, como se muestra en la figura 2.12.

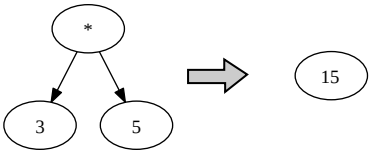


Figura 2.12: Multiplicación de constantes

Toda división de dos constantes produce una nueva constante, como se ejemplifica en la Figura 2.13.

Como protección contra las divisiones por cero, el sistema produce como resultado un cero, como se muestra en la Figura 2.14.

El absoluto de una constante produce otra constante, como se puede ver en la Figura 2.15.

Como protección contra la raíz cuadrada de valores negativos, el sistema produce un cero, como se ejemplifica en la Figura 2.16.

El *seno* de cualquier constante produce una nueva constante, como se muestra en la Figura 2.17.

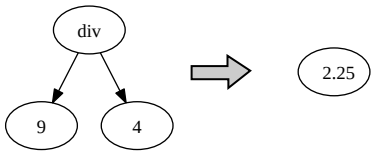


Figura 2.13: División de constantes

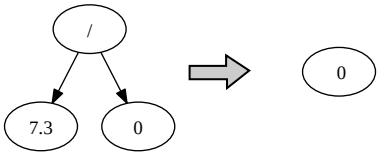


Figura 2.14: División por cero

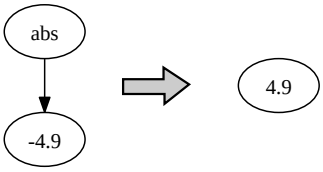


Figura 2.15: Absoluto de constante

El *cos* de cualquier constante genera otra constante, como se observa en la Figura 2.18.

La *tangente* de cualquier constante produce otra constante como se puede ver en la Figura 2.19.

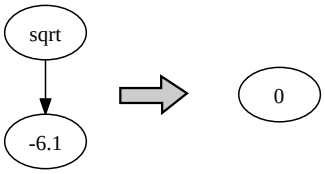


Figura 2.16: Raíz cuadrada negativa

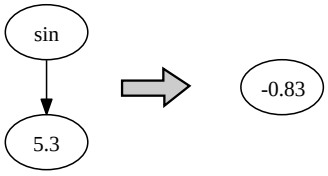


Figura 2.17: Seno de constante

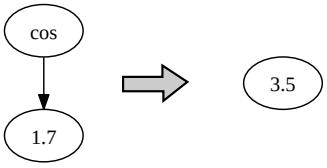


Figura 2.18: Coseno de constante

2.3.4. Encapsulación

El funcionamiento de este operador consiste en asignar un nombre a una parte de un árbol seleccionado de la población, convirtiendo dicha parte en una nueva *Terminal*. En la Figura 2.20 se ejemplifica su funcionamiento, donde el nodo sombreado

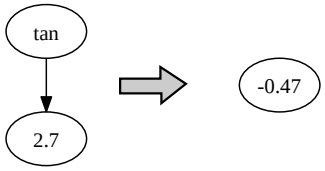


Figura 2.19: Tangente de constante

con valor + es seleccionado, por lo tanto, todo contenido debajo de él, incluido dicho nodo, será reemplazada por un único nodo que representará a todo el contenido removido, el cuál es una nueva terminal llamada ***NT1***. La ventaja de este operador es el poder convertir una rama en una hoja.

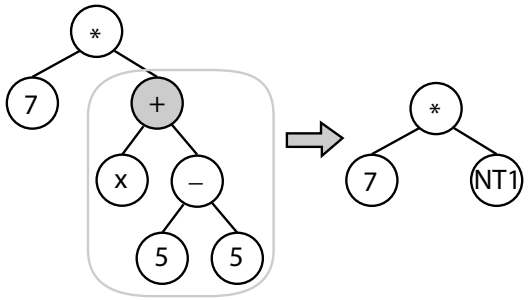


Figura 2.20: Encapsulación

2.4. Aptitud

La evaluación es una de las piezas clave en el tema de la evolución, debido a que ésta dirige el control evolutivo de los individuos, aportando el conocimiento para determinar si los individuos evaluados son aptos para la solución de un problema. Para esto es necesario una unidad de medida con la que se tenga la capacidad de medir o identificar la diferencia entre el comportamiento del modelo buscado contra

el comportamiento del modelo encontrado. A esta unidad de medida se le llama *aptitud* y la manera de obtener su valor no es única, por lo que para la evaluación de los individuos en esta tesis se utiliza la ecuación 2.1, que representa el negativo del error medio cuadrático (MSE) [Allen, 1971], dado que IGP esta maximizando la función objetivo.

$$aptitud = (-)\frac{\sum_i^N (f(x_i) - y_i)^2}{N} \tag{2.1}$$

Al realizar este cálculo se busca reducir el error del conjunto de observaciones $O = \{(x_1, y_1), (x_2, y_2)...(x_N, y_N)\}$. Siendo O el conjunto de N observaciones, donde N es la cantidad de observaciones que se tienen del sistema a modelar. y_i representan el comportamiento buscado, mientras que $f(x_i)$ representan el comportamiento encontrado.

En el Algoritmo 1 se observa que después de evaluar un individuo, el siguiente paso es insertarlo dentro de la población, por lo que el Algoritmo 6 muestra dicho proceso.

Algoritmo 6 Algoritmo para la inserción de individuos dentro de la población

```
INSERTAR(población, individuo)
1  peor_ind ← SELECCIÓN_TORNEO(población, peor)
2  población[peor_ind] ← individuo
```

Este algoritmo recibe dos parámetros, la *población* y el *individuo* a insertar. Como se puede observar en la línea 1, al igual que en el Algoritmo 5 se hace uso del Algoritmo *Selección_Torneo* con la diferencia en que en este momento se utiliza para conocer la ubicación del peor individuo, por lo que se realiza una selección que regrese el individuo que menor *aptitud* tenga. Una vez conocida dicha ubicación, como se

muestra en la línea 2, el *individuo* recibido como parámetro se inserta dentro de la posición que nos proporciona *Selección_Torneo*.

2.5. Selección

La selección es la manera de escoger a los individuos para después aplicarles algún operador, generando un nuevo individuo, el cual, mediante un criterio de inserción, podrá agregarse a la población. Existen diferentes formas para elegir a los individuos (ver [Whitley, 1989], [Blickle and Thiele, 1995], [Thierens and Goldberg, 1994], [Holland, 1992]), por ejemplo, se pueden seleccionar sólo los mejores individuos, seleccionar n individuos al azar escogiendo los mejores de ellos, o seleccionar usando la selección proporcional a la función de aptitud, donde todos los individuos tienen asignada una probabilidad de ser seleccionados.

Para este trabajo se ha utilizado la segunda opción, esto es, seleccionar n individuos al azar, escogiendo al mejor haciendo uso de la técnica conocida como *Selección por Torneo*, que se muestra en el Algoritmo 7 para el caso de $n = 2$. Usando esta técnica los n individuos son seleccionados para competir por ser los mejores o peores en base a su *aptitud*.

Este algoritmo recibe 2 valores como parámetro, el tamaño de la población y un valor booleano denominado *tipo*, el cual sólo puede tener como valor *true* o *false*. En la línea 1, al igual que en la 2, se selecciona de la población de manera aleatoria a un individuo haciendo uso del método *Entero_aleatorio*, el cuál retorna un valor aleatorio en 0 y el valor que recibe como parámetro, que corresponde a la posición de un individuo dentro de la población. En las líneas 3 y 4 se verifica que no se haya seleccionado el mismo individuo, si fue así, volverá a seleccionar otro de la misma manera antes mencionada. En la línea 5 se declara una condición que involucra al parámetro *tipo*, esta condición buscará al mejor o peor individuo según sea el valor de este parámetro. En las líneas 7 a 11 se regresa el individuo que peor *aptitud* contenga. Mientras que en las líneas 12 a 17 se regresa el individuo que mejor *aptitud* contenga.

Algoritmo 7 Selección mediante Torneo entre 2 individuos

```
SELECCIÓN_TORNEO(tam_población, tipo)
 1  ind_1 ← ENTERO_ALEATORIO(tam_población)
 2  ind_2 ← ENTERO_ALEATORIO(tam_población)
 3  mientras ind_1 == ind_2
 4    ind_2 ← ENTERO_ALEATORIO(tam_población)
 5  si tipo == peor
 6     entonces
 7                si aptitud[ind_1] < aptitud[ind_2]
 8                   entonces
 9                              regresar ind_1
10                   sino
11                              regresar ind_2
12     sino
13                si aptitud[ind_1] > aptitud[ind_2]
14                   entonces
15                              regresar ind_1
16                   sino
17                              regresar ind_2
```

2.6. Criterio de terminación

El criterio de terminación o condición de paro pretende identificar en que momento dejará de evolucionar la población. Ya se ha definido la *aptitud*, que nos ayuda a comparar dos comportamientos, por lo tanto, se pueden utilizar dos criterios de terminación:

- Definir una *aptitud*. En este caso el sistema continuará con la evolución de la población mientras el mejor individuo no cumpla con un valor de *aptitud* mínimo.
- Definir un número de generaciones. En este caso el sistema evolucionará la población mientras el número de generaciones no se haya cumplido.

Es común utilizar la primera opción cuando el problema es difícil de encontrar y por lo tanto, no se tiene idea del tiempo o generaciones que puedan ser necesarias para evolucionar la población y así poder resolver dicho problema.

Hasta ahora se ha explicado el funcionamiento de PG, así como sus operadores genéticos, los cuales son necesarios para realizar la evolución de una población. También se observó la necesidad de utilizar una unidad de medida que ayude a diferenciar la *aptitud* entre los individuos y finalmentde se describió el criterio de terminación, siendo este necesario para detener la evolución en un momento determinado. En el siguiente capítulo se tratará la estrategia propuesta en esta tesis para incorporar en un Algoritmo que haga uso de PG.

2.7. Software Relacionado

En la actualidad existe software tanto libre como comercial con la capacidad de realizar identificación de sistemas, como es el caso de *Eureqa* [Schmidt M., 2009], ECJ [White, 2012], lil-gp [Gagné and Parizeau, 2006] y BeagleGP [Wilson et al., 2004], entre otros. Para esta tesis se ha seleccionado el primero para comparar sus resultados con los obtenidos por el sistema desarrollado en este trabajo, ya que Eureqa ha sido probado para encontrar modelos de sistemas físicos.

2.7.1. Eureqa

Eureqa, a veces llamado robot científico [Keim, 2009b] es una herramienta de software comercial para la identificación de ecuaciones matemáticas [Keim, 2009a], creada por el Dr. Hod Lipson [Schmidt M., 2009] en el Computational Synthesis Lab en Cornell University. Este software ha sido creado para ser usado por expertos y no expertos en el área de PG, por lo que se ha utilizado para la solución de diferentes problemas, por ejemplo en [Dubčáková, 2011] fue usado para la detección de Radón en la República Checa, en [Srečec et al., 2013] usado para encontrar modelos matemáticos, así como en otros trabajos relacionados con procesamiento de imágenes como en [Allgaier and McDevitt, 2013].

El objetivo de *Eureqa* es utilizar técnicas avanzadas del área de computación evolutiva y regresión simbólica para identificar las más simples fórmulas matemáticas que podrían describir los mecanismos involucrados que producen los datos.

Esta herramienta cuenta con una versión de prueba de 30 días, la cual se puede descargar desde la siguiente dirección:

$http://creativemachines.cornell.edu/eureqa\_download$

O bien, obtener una version estudiantil para extender su uso sin ningún costo.

Su interfaz es muy intuitiva, por lo que resulta sencillo familiarizarse con ella y poderla utilizar de forma inmediata.

Para poder iniciar la búsqueda de un modelo usando *Eureqa*, principalmente se deben llevar a cabo 3 pasos, los cuales se ubican por pestañas dentro del programa, las cuales se describen a continuación:

- Inserción de datos.- Aquí se le proporcionan a *Eureqa* las observaciones que se tienen del sistema del cual se está interesado en encontrar su modelo.

- Selección del tipo de ecuación.- En este punto se determina el tipo de ecuación que se busca. Además, se seleccionan las funciones aritméticas, trigonométricas, entre otras, que se utilizarán para la construcción del modelo.

- Iniciar búsqueda.- En este momento *Eureqa* ya está listo para iniciar la búsqueda de la ecuación que más se acerque al comportamiento de los datos que se le

han proporcionado, utilizando las funciones antes mencionadas. Ahora sólo es necesario presionar el botón Run para que *Eureqa* inicie su trabajo.

Una vez que se inicia la búsqueda, el usuario tiene acceso a pausar, reanudar o detener *Eureqa*. Además, cuenta con otras dos pestañas cuyas funciones se describen en seguida:

- Ver resultados.- En esta sección se muestra en tiempo real los mejores modelos que hasta ese momento se han encontrado con su respectiva descripción, esto es, tamaño, *aptitud*, estructura del modelo, comportamiento del modelo de forma gráfica, entre otros datos.

- Reporte.- Aquí se muestra un reporte acerca de los modelos vistos en la sección anterior, con datos adicionales, como por ejemplo el tiempo fue necesario para obtener esos resultados. Este es el último paso necesario para conocer el modelo encontrado por *Eureqa* y puede ser guardado en diferentes formatos, como por ejemplo pdf.

2.8. conclusión

En este capítulo se ha visto la identificación de sistemas mediante PG, mostrando la manera en que PG trabaja, la generación de los individuos que representan una población, así como los principales operadores genéticos, recordando que para esta tesis sólo se han implementado tres, que son Cruza, Mutación subárbol y Mutación puntual. El operador Simplificación ha sido utilizado solo al momento de mostrar un modelo encontrado, pero no en el proceso de la evolución. También se mostró la unidad de medida *aptitud* utilizada para conocer las diferencias entre los modelos buscados con los encontrados. Después se explicaron dos criterios de terminación que podían utilizarse para definir la condición de paro. Se finalizó con la descripción de la herramienta de software llamada *Eureqa*, con quien se comparan los resultados obtenidos por el sistema IGP.

Capítulo 3

Evaluación de la aptitud por submuestreos incrementales

La evaluación de la aptitud por submuestreos incrementales es la estrategia para PG que se propone con el desarrollo de esta tesis. PG, al evolucionar su población mediante aprendizaje supervisado, necesita el conjunto de observaciones del sistema que se pretenda modelar. Normalmente se evalúa este conjunto de forma completa, por lo que en esta tesis se propone evaluarlas por submuestreos incrementales, es decir buscando modelos que poco a poco se ajusten al comportamiento buscado.

Para llegar a dicha estrategia fueron necesarias varias pruebas con otras estrategias pensadas en su momento, inicialmente se experimentó evaluando el conjunto de entrenamiento por completo. Se observó que en ocasiones la evolución era muy lenta ya que llegaba a un punto en donde la mejora del modelo encontrado era muy poca, por lo que se optó por buscar distintas maneras de realizar esta evaluación.

Las estrategias para seleccionar y evaluar los subconjuntos de las observaciones fueron las siguientes:

- Seleccionando submuestreos incrementales. En este caso, se seleccionan n observaciones seleccionadas de manera distribuida del conjunto total. Cuando se obtiene un modelo aceptable para dicho subconjunto, se procede con incrementar n, es decir, ahora seleccionar una cantidad de observaciones mayor a

la anterior, hasta terminar con el subconjunto del mismo tamaño al conjunto completo.

- Subconjuntos de izquierda a derecha. Esto es, seleccionando un subconjunto de observaciones de tamaño temporal, conformado por las primeras observaciones, donde, una vez encontrado el modelo para dicho subconjunto, se procede con incrementar su tamaño, agregando el siguiente subconjunto, hasta terminar evaluando el conjunto completo de observaciones del sistema.

- Subconjuntos de derecha a izquierda. Es similar al anterior, con la diferencia en que las observaciones se seleccionan del final hacia el inicio.

- Evaluando primero la parte que visualmente se consideraba más compleja. Para este caso, las señal se dividió en tres partes, considerando evaluar primeramente la que se observaba con mayor grado de complejidad, y seguido a esto, ir evaluando las demás partes conforme se encontraba el modelo para las observaciones que se acumulaban tras cada modelo encontrado.

De las estrategias antes mensionadas, IGP utiliza la primera opción, es decir, submuestreos incrementales, debido a que realizaron experimentos utilizando un conjunto de observaciones de N datos, siendo N el tamaño de las observaciones, para los cuales solo 5 ejecuciones para cada caso fueron suficientes para notar que la mejor opción para utilizarse de las antes mencionadas fue la primera, ya que se pudo observar que evaluar las observaciones de esta manera agilizaba el proceso de la evolución.

Otro dato a tomar en cuenta es el criterio a utilizar para la inserción de los individuos dentro de la población, los cuáles son generados por algún operador genético. Recordemos que IGP es un algoritmo de estado estable, por lo que en estas condiciones es posible insertar al nuevo individuo siguiendo los siguientes dos casos:

- Insertando en la población cualquier individuo. Esto se cumple cuando el individuo generado se inserta en la posición del individuo con menor *aptitud* seleccionado por el Algoritmo *Selección_Torneo*, sin tomar en cuenta su *aptitud*.

- Insertando en la población sólo los mejores individuos generados por los operadores genéticos utilizados. Esto se cumple cuando el individuo generado tiene mejor *aptitud* que el individuo seleccionado por el Algoritmo *Selección_Torneo* con peor *aptitud*. Se puede observar que con esta heurística se está incrementando la presión de selección.

Para los experimentos realizados con el sistema desarrollado se utilizó la segunda opción, ya que es la forma en que comúnmente se insertan los individuos en la población.

Ahora se explicará la selección de las observaciones de manera que se evalúen dichas observaciones por submuestreos incrementales. Para esto se propone el uso de subconjuntos, esto es $O_i = \{(x_1, y_1), (x_2, y_2)...(x_n, y_n)\}$, donde sus elementos son seleccionadas de forma distribuida, siendo $n < N$, $O_i \subseteq O$, y $|O_i| < |O|$, con $0 \leq i \leq \omega$, donde ω es el número de veces que se realizaran submuestreos con un nuevo tamaño de observaciones. De manera general, se pretende buscar el modelo para un número pequeño de observaciones, una vez encontrado, ese número de observaciones aumentará hasta que $|O_i| = |O|$.

A continuación se de definirá el *incremento*, que es un parámetro propio de la propuesta de evaluación del conjunto de observaciones que se presenta en esta tesis. Este parámetro tiene como objetivo determinar el número de observaciones a evaluar a través de la búsqueda del modelo de un sistema determinado.

Tomando en cuenta la Figura 3.1, dado un conjunto de 15 observaciones, en este caso el primer renglón de círculos con relleno gris corresponde al total de dichas observaciones. Usando un incremento de 4, el primer paso es seleccionar 4 elementos del total de observaciones, en este caso el dato 1, 5, 9 y 13, que se encuentran en el segundo renglón. Ahora se procede con la búsqueda del modelo para esos datos. Una vez encontrado dicho modelo nuevamente se selecciona un subconjunto del conjunto de observaciones, pero ahora será un total de 8, ya que inicialmente se seleccionaron 4, pero ahora se toma en cuenta el incremento, por lo tanto se seleccionan 8 observaciones, las cuales se muestran en el tercer renglón. Nuevamente se debe buscar un modelo aceptable para el nuevo subconjunto. Una vez encontrado el nuevo modelo, se

vuelven a seleccionar las observaciones, siendo ahora un total de 12 como se muestra en el cuarto renglón. Nuevamente se busca un modelo para el subconjunto de observaciones. Finalmente se termina por seleccionar el total de observaciones y buscar su modelo, como se muestra en el quinto renglón.

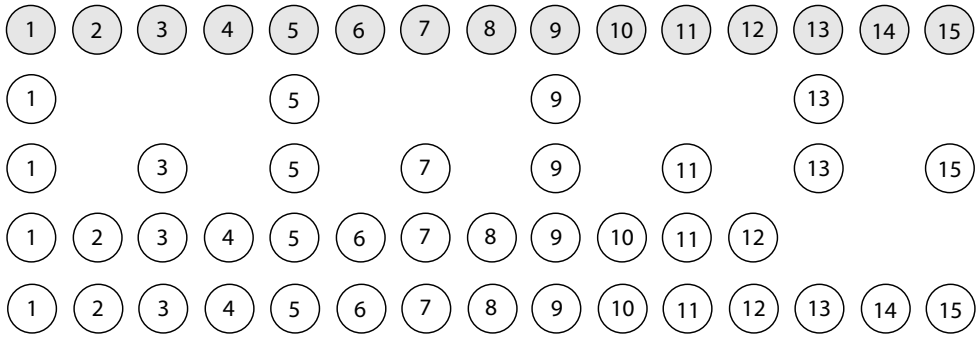


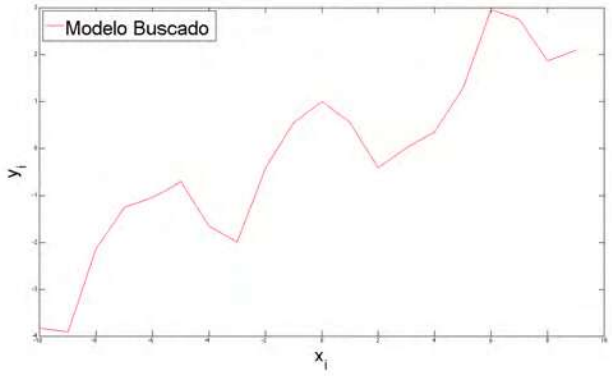
Figura 3.1: Selección de observaciones con incremento de 4.

A continuación se presenta un ejemplo utilizando el sistema IGP para reforzar la explicación de la selección propuesta. El tamaño de dichas observaciones(x y y) es de 20 elementos. En el inciso (a) de la Figura 3.2 se muestran el conjunto total de observaciones, estas representan el comportamiento del sistema real que se pretende modelar. mientras que en el inciso (b) se muestra dicho comportamiento graficado y en el inciso (c) se muestra la estructura de árbol que representa el comportamiento graficado.

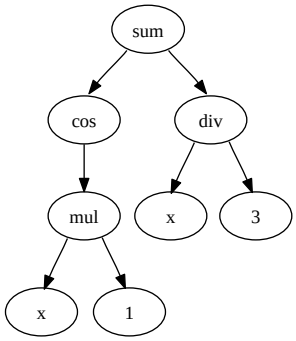
Una vez declarado el conjunto de observaciones y conociendo el comportamiento que deberá tener el modelo encontrado, se procede con submuestrear las observaciones, que como se puede observar en el inciso (a) de la Figura 3.3 las observaciones seleccionadas se encuentran marcadas por flechas, teniendo la misma distancia de separación entre ellas. Por lo que el subconjunto $O_1 = \{(-10, -3.83), (-6, -1.03), (-2, -0.41), (2, -0.41), (6, 2.96)\}$. Ahora IGP trata de encontrar un individuo que se comporte como O_i. En el inciso (c) de la Figura 3.3 se muestra el mejor individuo encontrado para las observaciones señaladas con las flechas del inciso (a), mientras que en inciso (b) se esta graficando el comportamiento buscado contra el comportamiento encontrado, donde se puede observar que el comportamiento hasta ahora encontrado no es muy

y_i	-3.83	-3.91	-2.14	-1.24	-1.03	-0.71	-1.65	-1.98	-0.41	-0.54	1.0	0.54	-0.41	0.01	0.34	1.28	2.96	2.75	1.85	2.08
x_i	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1	-0	1	2	3	4	5	6	7	8	9

(a)



(b)



(c)

Figura 3.2: Declaración y comportamiento del conjunto O.

bueno, y esto se debe a que el modelo ignora el 75 % de las observaciones que no han sido seleccionadas. La ventaja de tener pocas observaciones provoca que la evaluación sea mucho más rápida. Ahora se procede a seleccionar las observaciones siguientes.

El inciso (a) de la Figura 3.4 señala el subconjunto de observaciones $O_2 = \{(-10, -3.83), (-8, -2.14), (-6, -1.03), (-4, -1.65), (-2, -0.41), (0, 1), (2, -0.41), (4, 0.34), (6, 2.96), (8, 1.85)\}$. Esto es $|O_2| = 10$. En el inciso (c) de la Figura 3.4 se muestra el mejor individuo hasta el momento y en el (b) se puede observar que el comportamiento encontrado para este subuestreo ha mejorado, pero sigue teniendo problemas para tener el comportamiento buscado debido a que aún se ignora el 50 % de las observaciones. Por lo que se continúa submuestreando dichas observaciones.

Continuando con la selección de las observaciones, ahora $|O_3| = 15$, teniendo una distancia de cero en la separación entre las observaciones, además se mejora ligeramente el modelo encontrado como se puede ver en la Figura 3.5.

Finalmente al incrementar el tamaño de las observaciones, ahora $|O_4| = 20$, por lo que $|O_4| = |O|$, y como se puede observar en la Figura 3.6 el comportamiento encontrado no mejoró más.

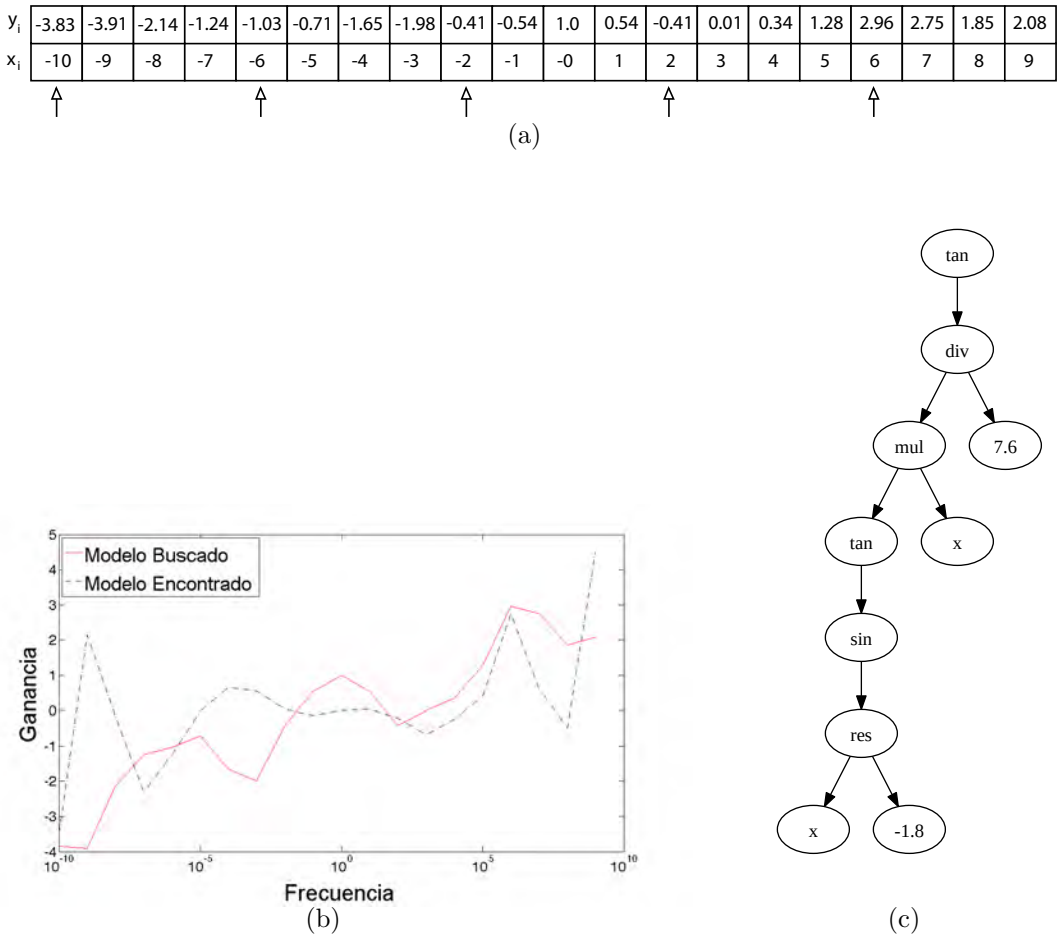


Figura 3.3: Selección del subconjunto O_1 y graficación del modelo encontrado.

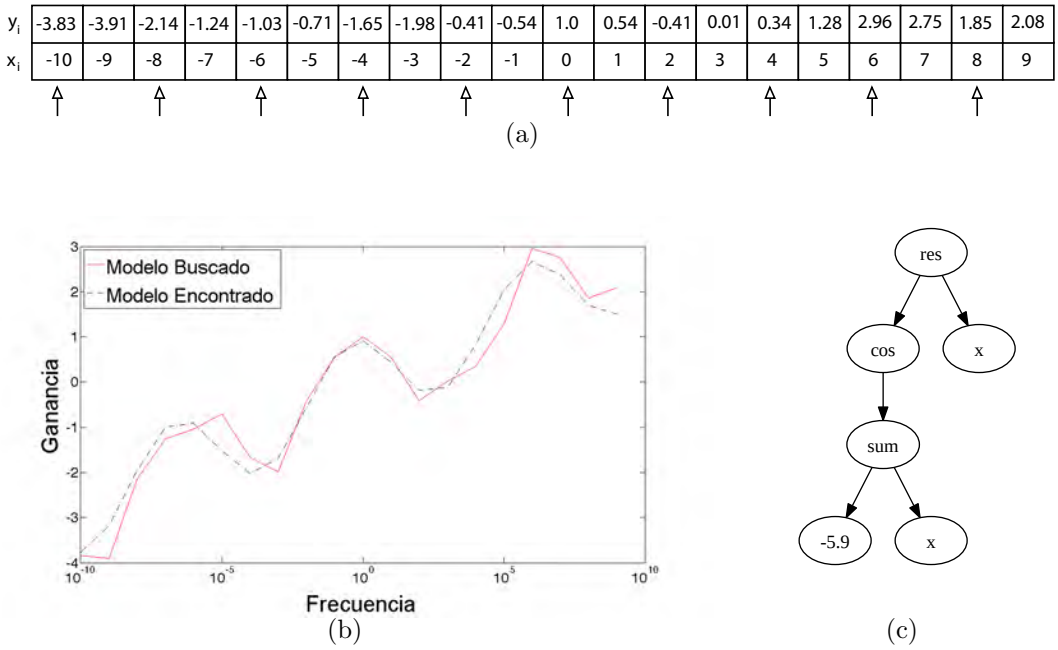


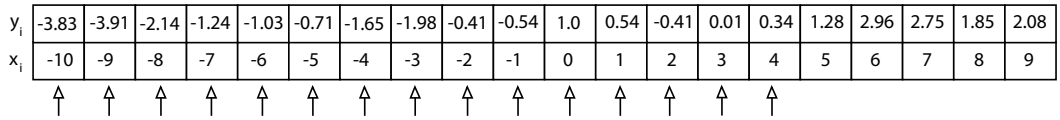
Figura 3.4: Selección del subconjunto O_2 y graficación del modelo encontrado.

3.1. Sistema IGP

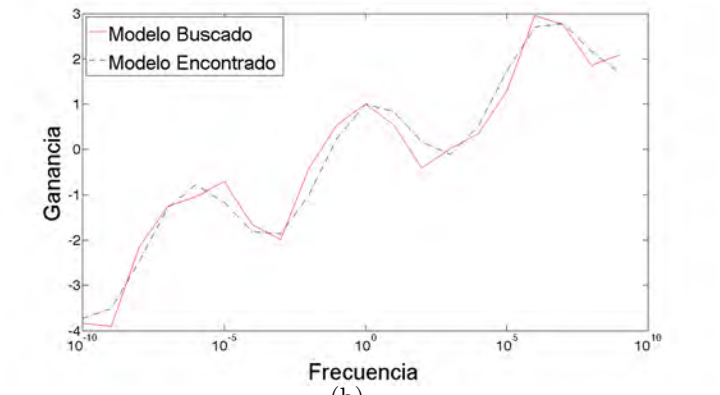
En este momento estamos en la posición de explicar el sistema de programación genética propuesto, llamado IGP. La Tabla 3.1 muestra los parámetros utilizados por esta propuesta para la realización de los experimentos del siguiente capítulo. De dicha tabla se pueden observar dos parámetros que no son tradicionales en PG, estos son, *costo máximo* e *incremento*.

El *Costo máximo* es una unidad de medida utilizada en los experimentos realizados en esta tesis. Fue necesaria su utilización debido a que en algunas ocasiones la evolución de los modelos era muy lenta, y por tal motivo se necesitaba detectar estos estancamientos e iniciar nuevas búsquedas de modelos. Su objetivo es finalizar la búsqueda de un modelo después de un número determinado de cálculos. En la ecuación 3.1 se muestra la manera de conocer su valor.

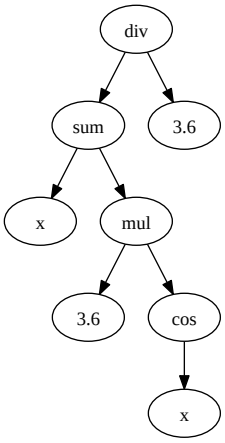
$$\Delta costo = \frac{n * t}{N * \alpha} \tag{3.1}$$



(a)



(b)



(c)

Figura 3.5: Selección del subconjunto O_3 y graficación del modelo encontrado.

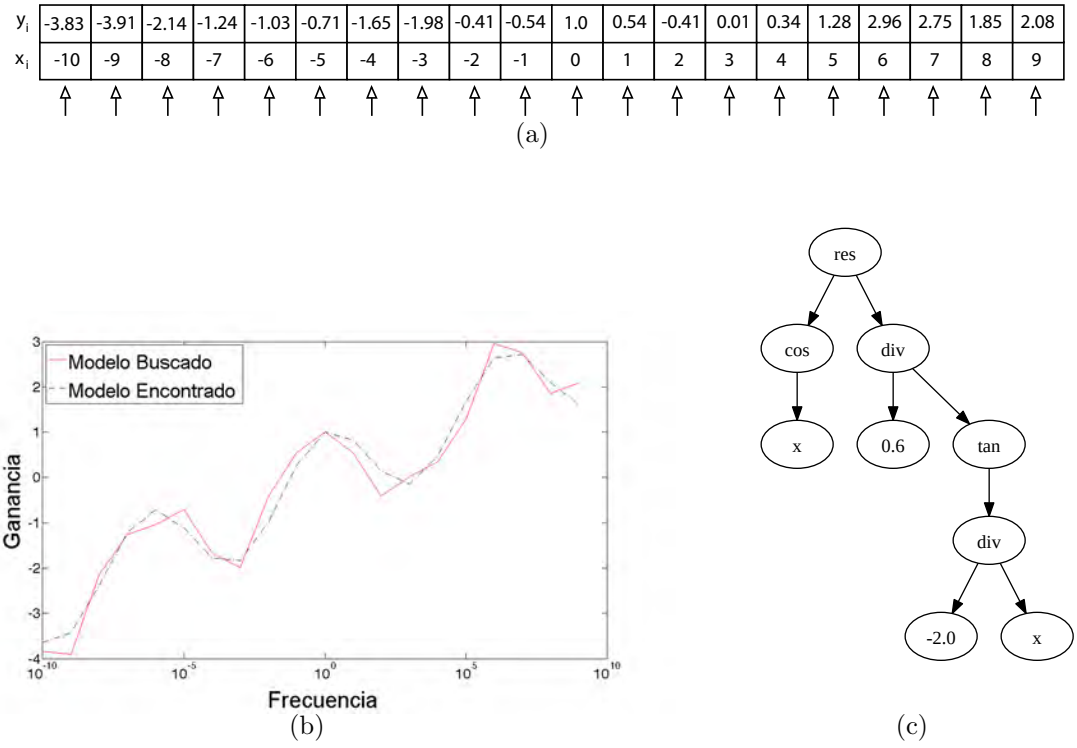


Figura 3.6: Selección del subconjunto O_4 y graficación del modelo encontrado.

Tabla 3.1: Datos iniciales

Parámetro	Valor
Conjunto de funciones	$\{+, -, *, /, abs, \sqrt{}, \sin, \cos, \tan\}$
Conjunto de terminales	$\{x,\ \mathbb{R}\}$
Constantes aleatorias (i.e., $\mathbb{R}$)	100 constantes $\in [-10, 10]$
Tamaño de la población	500
Tamaño máximo	2000
Probabilidad de cruza	90 %
Probabilidad de mutación	10 %
Costo máximo	$10,000$
Incremento	40

Como se puede observar en la ecuación 3.1, el costo es una variable que está en constante incremento y depende de cuatro variables, donde n es la cantidad de observaciones que se están evaluando en un momento determinado, N es el número total de observaciones que se tienen del sistema a modelar, t es el tamaño del individuo que se está evaluando, es decir, el número de nodos que lo conforma y finalmente α es un valor de escalamiento que representa a la cantidad $1,000,000$. Por lo tanto, el parámetro *Costo_Max* se utiliza para terminar la búsqueda del modelo cuando el *costo* llega a un valor límite.

El Algoritmo 8 presenta el pseudo-código para la implementación de IGP. En la línea 1 se generan las constantes utilizando el método *Genera_Constantes*, el cual recibe como parámetros el rango inicial, rango final y cantidad de constantes a generar. En la línea 2 se establecen las variables, pasando como parámetro la lista de variables a utilizar. En la línea 3 se procede con establecen las funciones, pasando como parámetro la lista de funciones utilizar. En la línea 4 se genera la población pasando como parámetro el número de individuos que la conformaran. En la línea 5 se genera una lista de aptitudes, correspondiente a cada uno de los individuos de la población. En la línea 6 se asigna a β el valor incremento, el cual es un parámetro de este algoritmo, para después, en la línea 7 establecer el valor de ω, el cual resulta de la división entre N que corresponde al número de observaciones e incremento, definido anteriormente. ω corresponde al número de veces que será necesario generar un nuevo submuestreo incremental. En la línea 9 se genera un ciclo que representa el recorrido que va desde

1 hasta el número de veces para submuestrear. En la línea 10 se submuestrea el conjunto de observaciones. En la siguiente línea, se observa la condición de paro. Esta condición se cumple cuando se logra obtener una *aptitud* deseable, la cual esta definida por la constante *APT_MIN* (aptitud mínima) o cuando *costo* cumple con un valor máximo definido por la constante *COST_MAX*. En la línea 13 se muestra el ciclo que va de 1 a N, esto representa una generación de nuevos individuos. En la línea 14 se genera un nuevo individuo llamado *hijo*, el cual resulta de la aplicación de los operadores genéticos utilizados por el sistema, para esto se hace uso del método *AplicaOperadorGenético* el cual se explica en el Algoritmo 5. En la línea siguiente se muestra la condición que controla el tamaño de los individuos para evitar que estos crezcan de manera incontrolable, permitiendo solo aquellos individuos que menores o iguales a un tamaño fijado por la constante *TAM_MAX*. En la línea 17 se selecciona al peor individuo haciendo uso del método *Selección_Torneo* el cual se explica en el Algoritmo 7. En la línea 18 - 20 se muestra la probabilidad de ocurrencia del 80 % para insertar el *hijo* en la población, en la ubicación del *peor* individuo obtenido anteriormente. Esta probabilidad esta definida por U, donde U(0,1) es un número generado aleatoriamente con una función de distribución uniforme entre 0 y 1. Si dicha condición no se cumple, el hijo podrá insertarse en la población solo si su *aptitud* es mejor que la *aptitud* del *peor* individuo seleccionado anteriormente, como se observa en las líneas 21 - 25.

Ahora se explicara el Algoritmo 9 utilizado por IGP para el submuestreo incremental de las observaciones. Donde dicho algoritmo recibe cuatro parámetros, i que corresponde a las veces que se han submuestreado las observaciones, ω que corresponde al número de veces que se deberán submuestrar, las *observaciones* del sistema a modelar y β que corresponde al incremento.

En la línea 1 se asignan las observaciones a y. En la línea 2 se asigna a N el valor correspondiente al número de observaciones. En la línea 3 se asigna a m el tamaño del submuestreo. En la línea 4 se asigna a S la separación que tendrán las observaciones seleccionadas. En la línea 5 se muestra la selección de x con m observaciones con una separación de S. Y finalmente, en la línea 6 se retorna dicha selección.

Una vez explicada la propuesta para la selección de los subconjuntos de observa-

Algoritmo 8 Funcionamiento de IGP

```
SISTEMA(observaciones, incremento)
 1  constantes ← GENERA_CONSTANTES(rango_inicial, rango_final, cantidad)
 2  variables ← ESTABLECE_VARIABLES(lista_variables)
 3  funciones ← ESTABLECE_FUNCIONES(lista_funciones)
 4  población ← CREA_POBLACIÓN(número_individuos)
 5  aptitudes ← EVALÚA_POBLACIÓN(población)
 6  β ← incremento
 7  ω ←N/β
 8  i = 1
 9  mientras i ≤ ω
10    selección ← SUBMUESTREO(i, ω, observaciones, β)
11    mientras aptitudes[mejor] < APT_MIN y costo < COST_MAX
12      j = 1
13      mientras j ≤ N
14        hijo ← APLICAOPERADORGENÉTICO(población)
15        si TAMAÑO(hijo) ≤ TAM_MAX
16           entonces
17                    peor = SELECCIÓN_TORNEO(población, mejor)
18                    si U(0,1) < 0.8
19                       entonces
20                                población[peor] ← hijo
21                       sino
22                                aptitud ← OBTEN_APTITUD(hijo)
23                                si aptitudes[peor] < aptitud
24                                   entonces
25                                             población[peor] ← hijo
26        j++
27    i++
```

Algoritmo 9 Selección de puntos de manera distribuida

SUBMUESTREO$(i, \omega, observaciones, \beta)$

1 $y \leftarrow observaciones$
2 $N \leftarrow |y|$
3 $m \leftarrow (i+1) * \beta$
4 $S \leftarrow N/m$
5 $x[m] \leftarrow y[mS]$
6 **regresar** x

ciones y continuando con la explicación del algoritmo que utiliza el sistema de PG desarrollado, ahora en el Algoritmo 10 se explican las instrucciones que se utilizan para realizar la *Cruza* entre dos individuos.

En la línea 1 y 2 se selecciona a *padre1* y *padre2* usando selección por torneo. En la línea 3 se selecciona de manera aleatoria a un nodo del *padre1*. De esta misma manera, en la línea 4 se selecciona otro nodo del *padre2*. Recordemos que los individuos son estructuras de árbol, por lo que el en la línea 5 se asigna a *subárbol* la rama de *padre2* que contiene el nodo *p2*. En la línea 6 se asigna a *hijo* la concatenación de *padre1* con *sub_árbol*, insertando la raíz del *sub_árbol* en la posición correspondiente al nodo *p1*. En la línea 7 se regresa el hijo generado.

En el Algoritmo 11 se muestran la Mutación para generar un nuevo individuo.

Este algoritmo, a diferencia con el anterior, sólo necesita de un individuo de la población para generar un nuevo individuo llamado hijo, además tiene dos modalidades o tipos de *Mutación*, la *Mutación_puntual* y *Mutación_subárbol*. Primeramente se selecciona el *padre1* de la población. Enseguida, en la línea 2, se decide que tipo de mutación se realizará, cada uno tiene la misma probabilidad. En la *Mutación_puntual*, se recorre al *padre1*, como se muestra en la línea 4. En la línea 5 se selecciona el nodo correspondiente al recorrido, asignando su valor a *p*, en seguida, en la línea 6 se comprueba si el nodo seleccionado es o no una *función*, si lo es, existe un 50 % de

Algoritmo 10 Operador Genético Cruza

```
CRUZA(población)
1  padre1 ← SELECCIÓN_TORNEO(población, mejor)
2  padre2 ← SELECCIÓN_TORNEO(población, mejor)
3  p1 ← ENTERO_ALEATORIO(|padre1|)
4  p2 ← ENTERO_ALEATORIO(|padre2|)
5  sub_árbol = OBTENPARTE(padre2, p2)
6  hijo ← CONCATENA(padre1, sub_árbol, p1)
7  regresar hijo
```

probabilidad de ser modificado, como se muestra en las líneas 9 a 11. Si el nodo seleccionado de *padre1* no es *función*, entonces es una *terminal*, por lo que existe una probabilidad del 50 % como se muestra en las línea 14 y 15, de que dicha *terminal* sea modificada por otra *terminal*, la cual puede ser una *constante* o una *variable*, y como se muestra en las líneas 17 y 18 primero se selecciona una terminal al azar para después reemplazar el nodo seleccionado del *padre1*. Si la condición inicial de la línea 2 no se cumple, entonces se realiza una *Mutación_subárbol*. Para realizar este tipo de *Mutación* es necesario generar un padre de manera aleatoria, por lo que en la línea 22 existe la probabilidad del 50 % de que el padre generado sea un *árbol_full* como se muestra en las líneas 23 y 24. Y la misma probabilidad para generar un *árbol_grow* como se muestra en las líneas 25 y 26. Al final, en la línea 28 se retorna el hijo que haya producido este algoritmo.

3.2. Conclusión

En este capítulo se ha visto la estrategia propuesta con esta tesis, que consiste en evaluar la *aptitud* en PG por submuestreos incrementales, la cual es implementada en el sistema IGP para la búsqueda de modelos, explicando con una figura y un

Algoritmo 11 Operador Genético Mutación

```
MUTACIÓN(población)
 1  padre1 ← SELECCIÓN_TORNEO(población, mejor)
 2  si U(0,1) < 0.5
 3     entonces
 4              para i ∈ padre1
 5                p ← padre1[i]
 6                si esFUNCIÓN(p)
 7                   entonces
 8                            si U(0,1) < 0.5
 9                               entonces
10                                        func ← funciónAZAR(p)
11                                        hijo ← AGREGA(hijo, func)
12                               sino
13                                        hijo ← AGREGA(hijo, p)
14                   sino
15                            si U(0,1) < 0.5
16                               entonces
17                                        term ← terminalAZAR
18                                        hijo ← AGREGA(hijo, term)
19                               sino
20                                        hijo ← AGREGA(hijo, p)
21     sino
22              si U(0,1) < 0.5
23                 entonces
24                         tmp ← ÁRBOL_FULL(U(1,5))
25                 sino
26                         tmp ← ÁRBOL_GROW(U(1,5))
27              hijo ← CRUZA(padre1, tmp)
28  regresar hijo
```

ejemplo, la manera de seleccionar las observaciones. También se mostró la Tabla 3.1 con los parámetros utilizados por IGP y finalmente se presentó el Algoritmo de la implementación de dicho sistema.

Capítulo 4

Resultados

En este capítulo se presentan los resultados obtenidos por el sistema IGP, siendo comparados contra los resultados obtenidos por el software *Eureqa*. Además se comprueba el funcionamiento de la estrategia propuesta haciendo uso de un total de nueve diferentes muestras de transformadores de potencia.

4.1. Análisis de datos iniciales

Los datos iniciales son los parámetros que nos permiten inicializar un sistema que hace uso de PG, por ejemplo, el *conjunto de funciones*, *conjunto de constantes* y *número de individuos* que conforman la población, entre otros. Todos estos son datos que pueden ser modificados a gusto del usuario o programador.

En la Tabla 3.1 del capítulo anterior se muestran los datos iniciales utilizados por el sistema IGP para la realización de los experimentos.

El *conjunto de funciones* es el conjunto de operadores aritméticos y trigonométricos que se emplearán para la búsqueda del modelo. Para definir este conjunto de funciones se realizaron 10 experimentos con la señal 1, para la cual se muestra su comportamiento en el capítulo 4. En dichos experimentos se probaron dos conjuntos de funciones:

- Utilizando el conjunto de funciones $\{+, -, *, /, sqrt, sin, cos, tan\}$ con la misma probabilidad de seleccionar cualquier elemento del conjunto.

- Utilizando el mismo conjunto anterior, con la diferencia en que la probabilidad de *sin*, *cos* y *tan* con el doble de probabilidad de selección a los otros elementos del conjunto.

Donde los mejores resultados se observaron utilizando el primer caso, esto es, que todas las funciones tubieran la misma probabilidad de ocurrir.

El *conjunto de terminales* está conformado por variables y constantes, las cuales conforman las hojas de los árboles generados en PG. Para los experimentos de este trabajo se utilizó una sola variable y un conjunto de 100 constantes generadas al azar, con valor real aleatorio entre -10 y 10.

El *tamaño de la población* corresponde al número de individuos que se crearán y mantendrán durante la evolución del sistema a través del uso de los operadores genéticos. En este trabajo se experimentó con:

- 50 ejecuciones con población de 1000 individuos
- 50 ejecuciones con población de 500 individuos

Donde los resultados obtenidos no mostraron diferencias notables con el uso de menor o mayor tamaño en la población, excepto en el consumo de memoria, ya que entre mayor sea dicho tamaño, mayor será el consumo de memoria en el equipo en que se pruebe el sistema.

El parámetro *tamaño máximo*, se refiere al tamaño que pueden llegar a tener los individuos. Este punto es muy importante debido a que se debe prestar atención al crecimiento de los individuos, ya que es muy fácil que dichos individuos aumenten su tamaño de forma muy rápida, lo que puede llegar a agotar la memoria disponible del equipo en el que se estén llevando a cabo los experimentos. En este trabajo se experimentó con tres diferentes tamaños máximos:

- 10 ejecuciones con tamaño máximo de 200 nodos
- 50 ejecuciones con tamaño máximo de 1000 nodos
- 50 ejecuciones con tamaño máximo de 2000 nodos

En la Figura 4.1 se muestra el mejor resultado obtenido al restringir el tamaño máximo de nodos con *Tam_Max*= 200. En esta figura se muestra de color rojo el comportamiento del modelo buscado, mientras que de color negro y punteado se muestra el comportamiento del modelo encontrado. Como se puede observar su *aptitud* es de -4.87 debido a que se le dificulta apegarse al comportamiento del modelo buscado. Las partes donde se aprecian mejor las diferencias son cerca del valor en $x = 10^3$, $x = 10^4$ y $x = 10^6$.

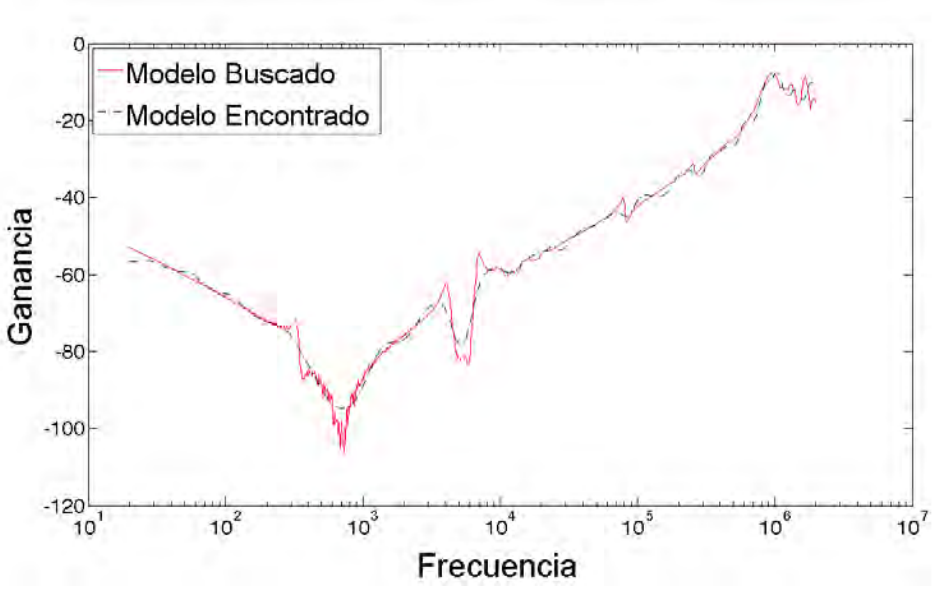


Figura 4.1: Mejor individuo con 191 nodos, *aptitud*= -4.87

A continuación, en la Figura 4.2 se muestra el mejor resultado obtenido, limitando el tamaño máximo de nodos con *Tam_Max*= 1000. Al igual que en la figura anterior, de color rojo se muestra el comportamiento del modelo buscado, mientras que de color negro y punteado se muestra el comportamiento del modelo encontrado. En esta figura a diferencia de la anterior, se puede notar un mejor comportamiento, ya que en las esquinas ubicadas en $x = 10^3$ y $x = 10^4$ el comportamiento es muy similar al esperado, aunque al final de la señal sigue teniendo problemas para imitar el comportamiento.

Ahora, en la Figura 4.3 se muestra el mejor resultado obtenido, limitando el tamaño de nodos con *Tam_Max*= 2000. En esta figura se puede observar que se mantiene

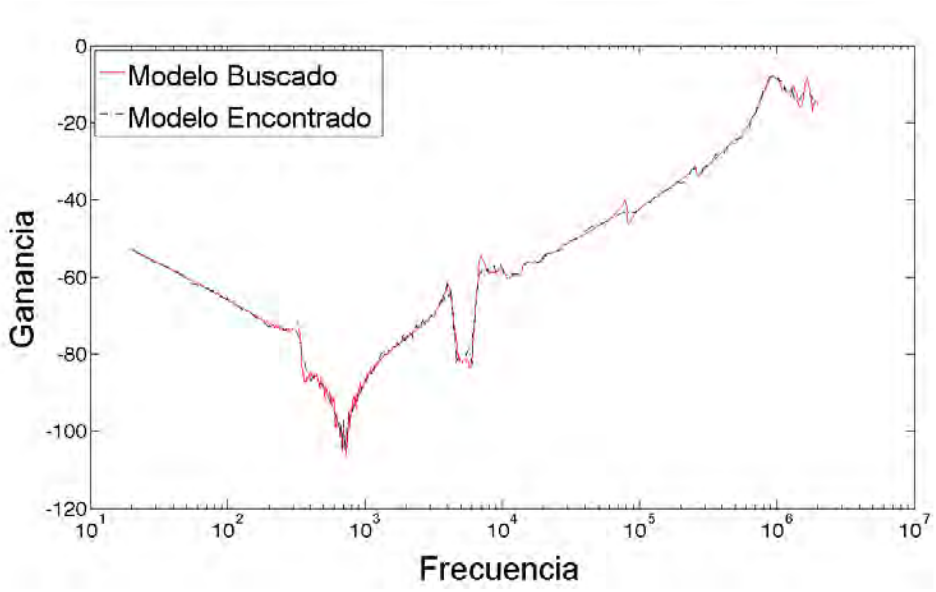


Figura 4.2: Mejor individuo con 844 nodos, *aptitud*= -1.12

en las esquinas con valor $x = 10^3$ y $x = 10^4$ se sigue apegando el comportamiento del modelo encontrado al modelo buscado, además en la parte final de la señal el comportamiento es mejor, apegándose lo suficiente al modelo buscado.

Con estos experimentos se logró entender que para este caso en particular, limitar el tamaño de los individuos puede afectar la evolución de los mismos, como ocurrió al utilizar un límite de 200. Para el caso del límite de 1000, el resultado no se ve tan mal, aunque con un límite de 2000 el mejor resultado obtenido fue mejor. Por lo que se concluye que entre menor sea el tamaño de nodos en un individuo, menor es el espacio de búsqueda y mayor es la dificultad para encontrar un buen modelo que simule el comportamiento buscado.

Como se mencionó en el capítulo anterior, los operadores genéticos son lo encargados de controlar la evolución de los individuos, por lo tanto no pueden faltar en un sistema que hace uso de PG. En este trabajo se hace uso solamente de la *cruza* y *mutación* debido a que comúnmente son los más usados en esta área, utilizando 90 % de *probabilidad de cruza* y 10 % de *probabilidad de mutación*. Para decidir esta probabilidad se realizaron 20 corridas con probabilidades de 90 % en cruza con 10 %

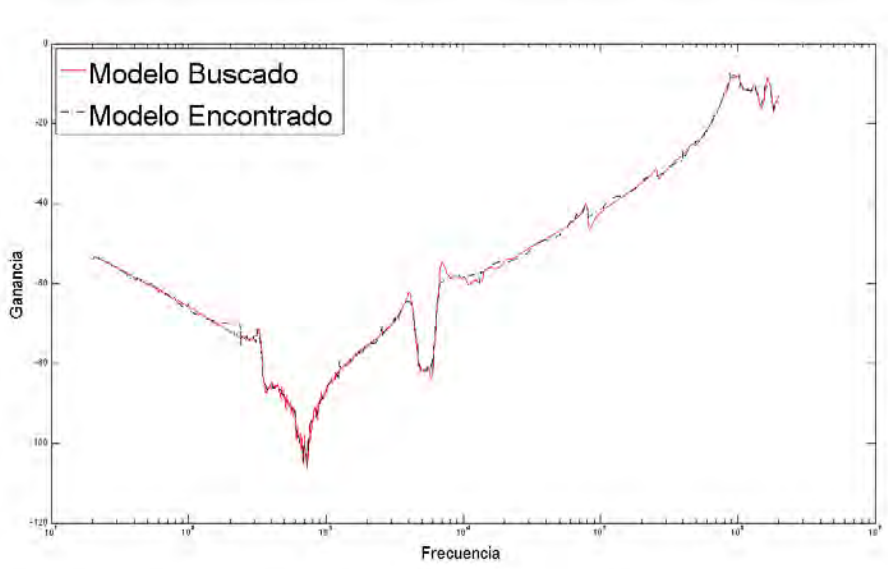


Figura 4.3: Mejor individuo con 1648 nodos, *aptitud*= -1.04

en mutación, además 20 corridas con probabilidad de 50 % para cruza y 50 % para mutación, obteniendo en ambos caso resultados similares, por lo que se decidió utilizar las probabilidades 90 %, 10 % ya que se pudo observar que variar estas probabilidades no afecta en gran medida el desarrollo de la evolución.

Tomando en cuenta los resultados, se puede observar que son operadores suficientes en la búsqueda de un modelo. Además en [Spears et al., 1992], [White and Poulding, 2009], [Luke and Spector, 1998] y [Luke and Spector, 1997] se demuestra mediante distintas pruebas que la diferencia entre usar mayores o menores probabilidades es insignificante, esto por que la Mutación obtiene ligeramente mejores resultados ante poblaciones pequeñas, mientras que la Cruza obtiene mejores resultados de manera general.

4.2. Regresión Simbólica

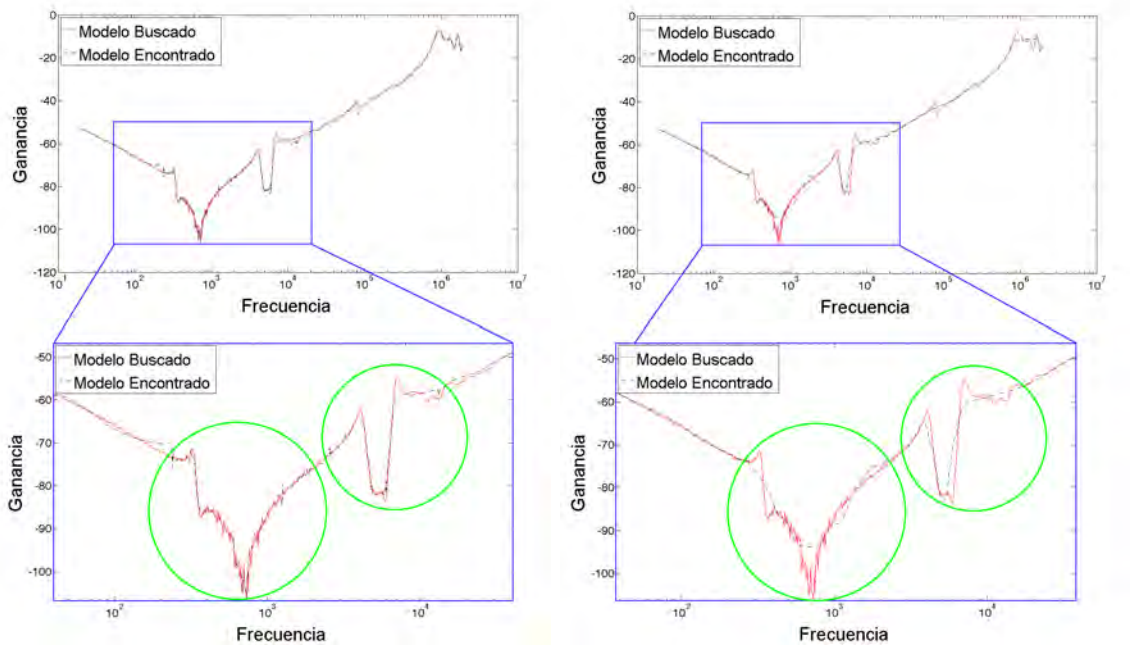
La regresión simbólica [Augusto and Barbosa, 2000] es una aplicación de la PG, donde dado el conjunto de observaciones de un sistema (datos) se busca una expresión matemática (patrón) que identifique el comportamiento de dicho sistema, accediendo

a todo tipo de funciones y combinaciones algebraicas como le sea posible. Es importante recalcar que los sistemas de prueba utilizados en este trabajo, como ya se mencionó, corresponden a la respuesta a la frecuencia de transformadores de potencia. Este tipo de dispositivos, como cualquier tipo de estructura magnética, presentan un comportamiento altamente no lineal lo cual dificulta la posibilidad de encontrar un modelo matemático que lo represente.

A continuación se analizan los resultados obtenidos de las nueve señales, comparando el sistema IGP contra *Eureqa*.

4.2.1. Señal 1

En la Figura 4.4 se puede observar el comportamiento del mejor modelo encontrado por IGP contra el mejor modelo encontrado por *Eureqa*, realizando un acercamiento a la primer mitad de la graficación de los resultados para ambos casos.



(a) Modelo obtenido por IGP, *aptitud*= -1.04. (b) Modelo obtenido por *Eureqa*, *aptitud*= -2.38.

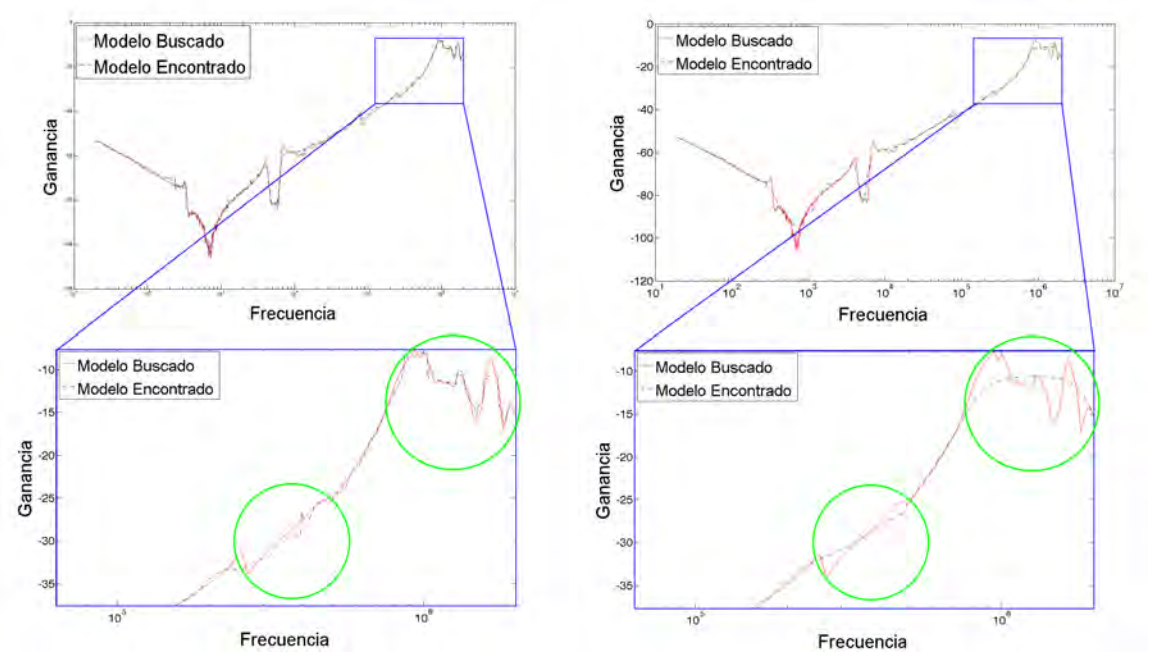
Figura 4.4: Mejores modelos obtenidos para la señal 1.

El modelo encontrado por IGP se apega mejor al comportamiento del modelo

buscado, ya que en las dos puntas mayor pronunciadas su comportamiento fue muy satisfactorio, aunque en algunas zonas faltó mayor precisión. Mientras que en el caso de *Eureqa* se observa un resulta menos aceptable en esas mismas zonas.

Cabe mencionar que el modelo encontrado por *Eureqa* tubo una duración de 5 días obteniendo una *aptitud* de -2.38, mientras que el modelo encontrado por IGP fue de 3 días con una *aptitud* de -1.04.

A continuación, en la Figura 4.5 se continúa con la comparación entre *Eureqa* e IGP, ahora mostrando un acercamiento a la parte final del modelo buscado para la señal 1.



(a) Modelo obtenido por IGP, *aptitud*= -1.04. (b) Modelo obtenido por *Eureqa*, *aptitud*= -2.38.

Figura 4.5: Mejores modelos obtenidos para la señal 1.

Como se puede observar en esta figura, de nuevo los mejores resultados se los queda IGP, ya que como se muestra en la parte central y final de este acercamiento, IGP se apega mejor al comportamiento buscado, mientras que a *Eureqa* se le sigue dificultando.

En seguida se presenta la Tabla 4.1, donde se muestra la media y desviación

estándar de una serie de experimentos entre *Eureqa*, PG evaluando la señal completa e IGP.

Tabla 4.1: Tabla de Aptitudes

Sistema	corridas	sin terminar	μ	σ
Eureqa	26	26	-7.83	-2.55
Toda la señal (*Tam_Max*=2000)	33	13	-4.34	-3.84
IGP (*Tam_Max*=2000)	33	4	-2.90	-1.81

Para dichos experimentos se fijó a *Apt_Min* en -2.3, por lo que todo modelo con una *aptitud* menor a este valor se considera que no logró encontrar un modelo aceptable. Y como se puede observar, el promedio de la *aptitud* de los resultados de IGP están en -2.90, superando a PG evaluando toda la señal, al igual que para el caso de la desviación estándar, IGP logra también un mejor resultado.

Ahora se muestra la Tabla 4.2, donde se presenta la media y *costo* de los experimentos de la Tabla 4.1, marcando las columnas de *Eureqa* con valor Desconocido, ya que no se tiene acceso a esa información.

Tabla 4.2: Tabla de Costos

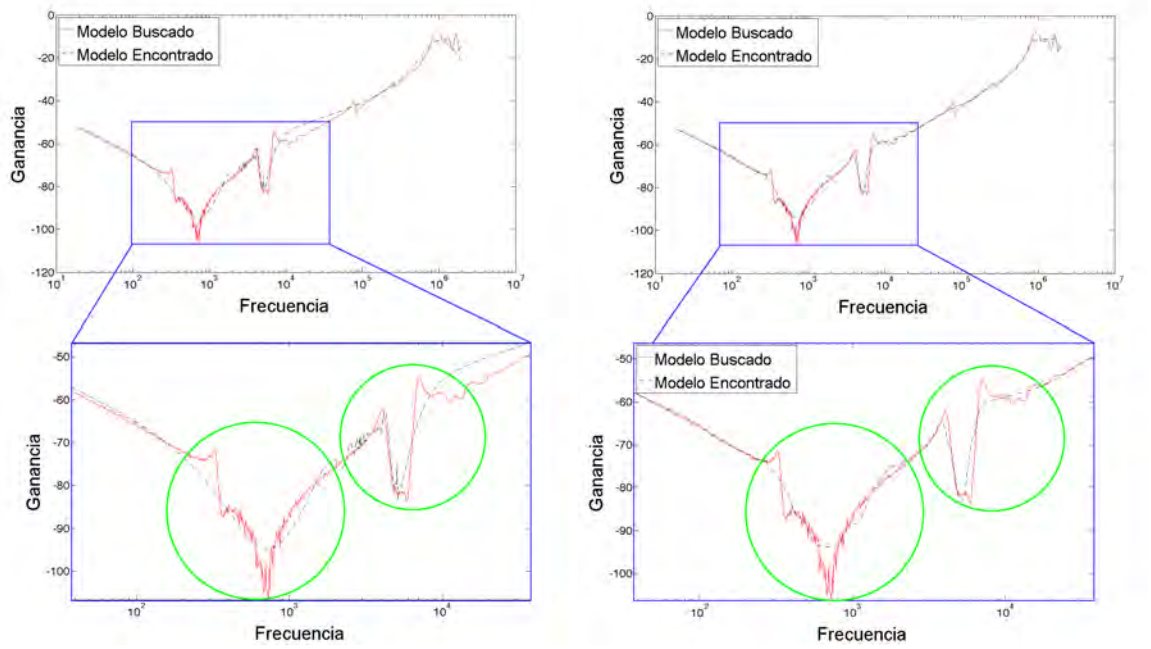
Sistema	μ	σ
Eureqa	Desconocido	Desconocido
Toda la señal (*Tam_Max*=2000)	5413	4204
IGP (*Tam_Max*=2000)	3688	3124

A continuación, en la Figura 4.6 se presenta la comparación entre el mejor modelo obtenido por IGP con el parámetro *Tam_Max*=39 y *Eureqa*.

En dicha figura se puede observar que al disminuir el tamaño máximo de los individuos en el sistema IGP, provoca que la solución encontrada no se apegue al comportamiento deseado, por lo que en el acercamiento de la 4.6 se puede observar que *Eureqa* obtiene mejores resultados que IGP.

En la Figura 4.7 se muestra ahora un acercamiento a la parte final de los modelos encontrados por IGP y *Eureqa*, observando que también en esta zona el ajuste obtenido por *Eureqa* ha sido mejor que IGP con el parámetro *Tam_Max*=39.

Como se mencionó en capítulos anteriores, los modelos obtenidos con IGP al igual que con *Eureqa* pueden ser representados con estructuras de árbol, por lo que a



(a) Modelo obtenido por IGP con *Tam_Max=39*, (b) Modelo obtenido por *Eureqa*, *aptitud*= -2.38. *aptitud*= -9.41.

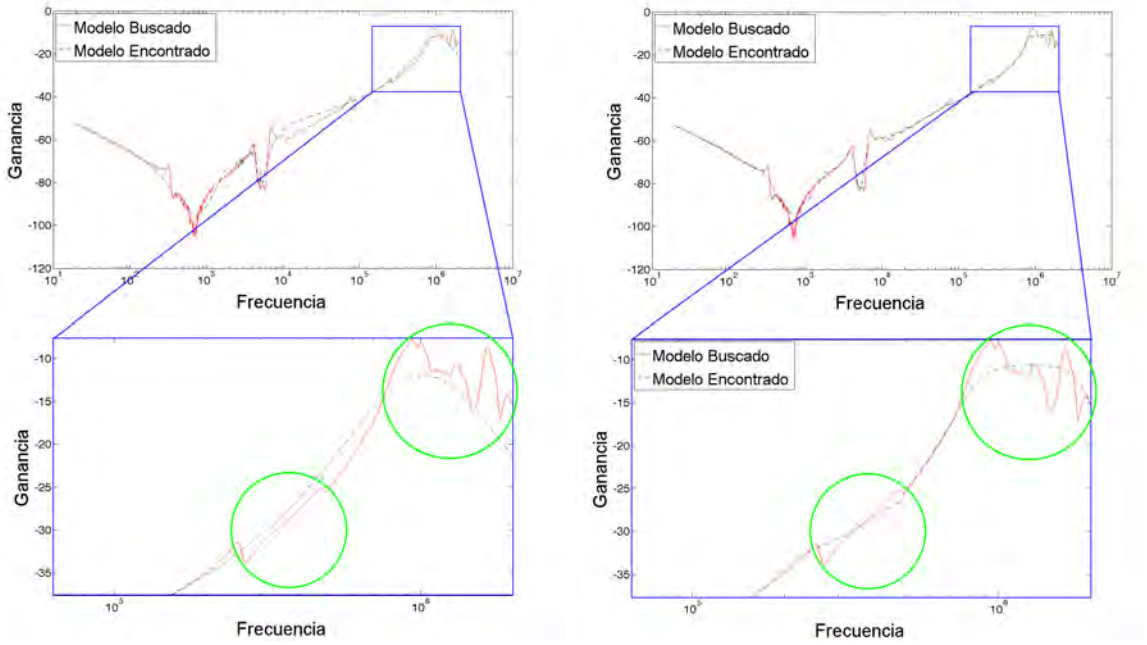
Figura 4.6: Mejores modelos obtenidos para la señal 1.

continuación se muestra los árboles que representan a los mejores modelos encontrados para la señal 1. La Figura 4.8 muestra el árbol generado por *Eureqa* que corresponde al mejor modelo encontrado y que fue mostrado en la Figura 4.4. Como se puede observar, su estructura es pequeña, comparada con las estructuras obtenidas por PG e IGP.

En la Figura 4.9 se muestra el árbol del mejor modelo encontrado por IGP, limitando su tamaño a 200 nodos, por lo que el tamaño de este árbol es de 191 nodos.

En la Figura 4.10 se muestra el árbol generado por el mejor modelo encontrado por IGP, como se puede observar es muy grande, e imposible de interpretar ya que cuenta con un total de 699 nodos, pero su comportamiento es muy cercano al comportamiento real del sistema, aspecto vital para intentar la aplicación de estas técnicas a la detección de fallas incipientes en transformadores de potencia.

En la Figura 4.11 se observa el árbol generado por el mejor modelos obtenido con



(a) Modelo obtenido por IGP con *Tam_Max=39*, (b) Modelo obtenido por *Eureqa*, *aptitud*= -2.38. *aptitud*= -9.41.

Figura 4.7: Mejores modelos obtenidos para la señal 1.

IGP limitando su tamaño a 2000, por lo que el tamaño de este árbol es de 1648 nodos.

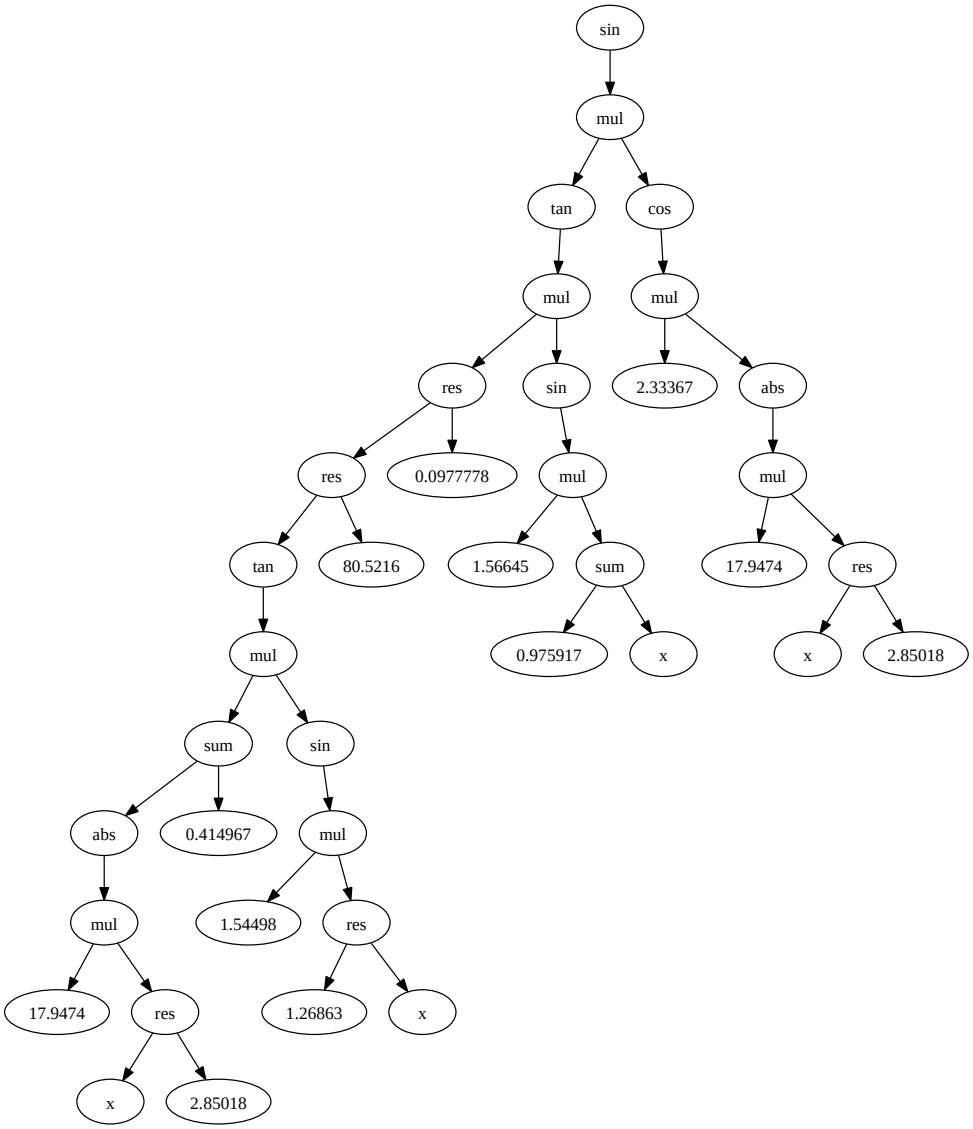


Figura 4.8: Árbol generado por *Eureqa* para la señal 1, 39 nodos

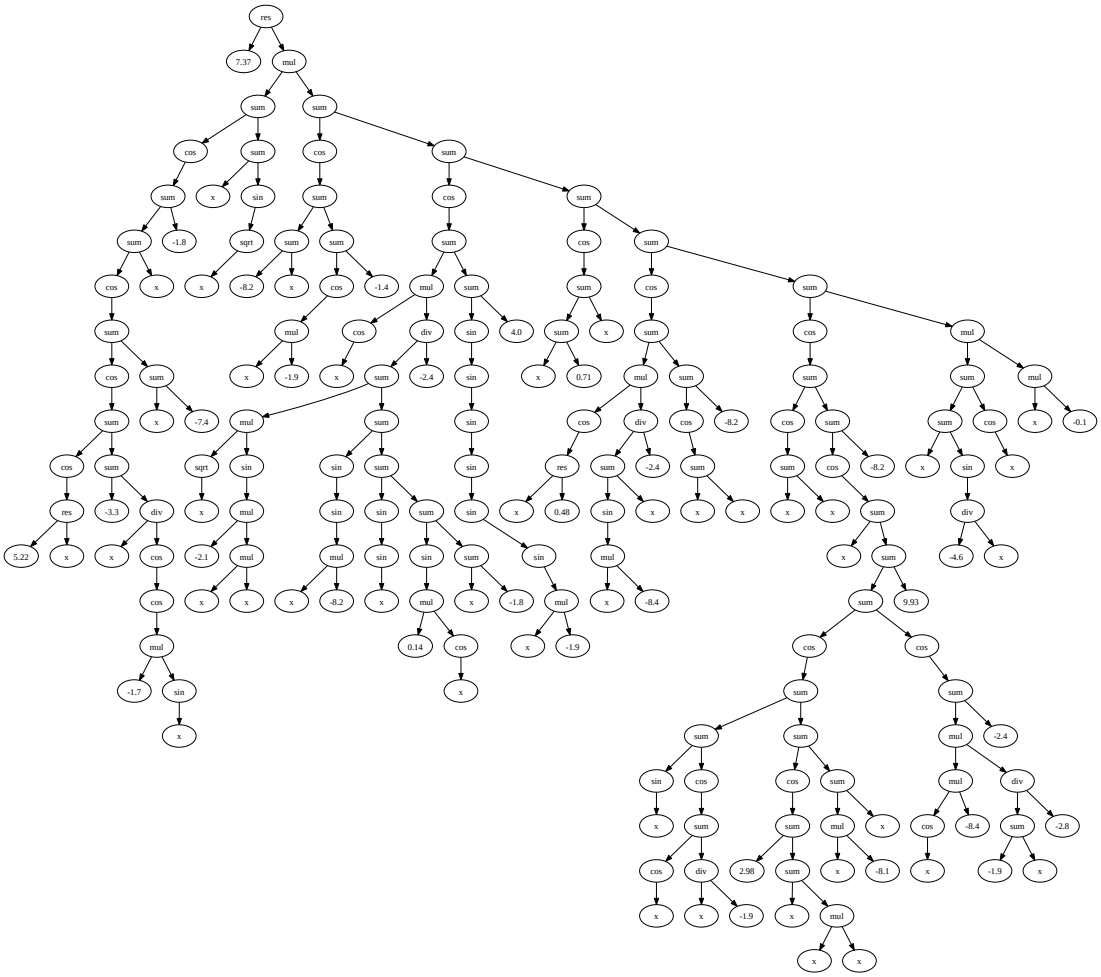


Figura 4.9: Árbol generado por IGP para la señal 1 con 191 nodos

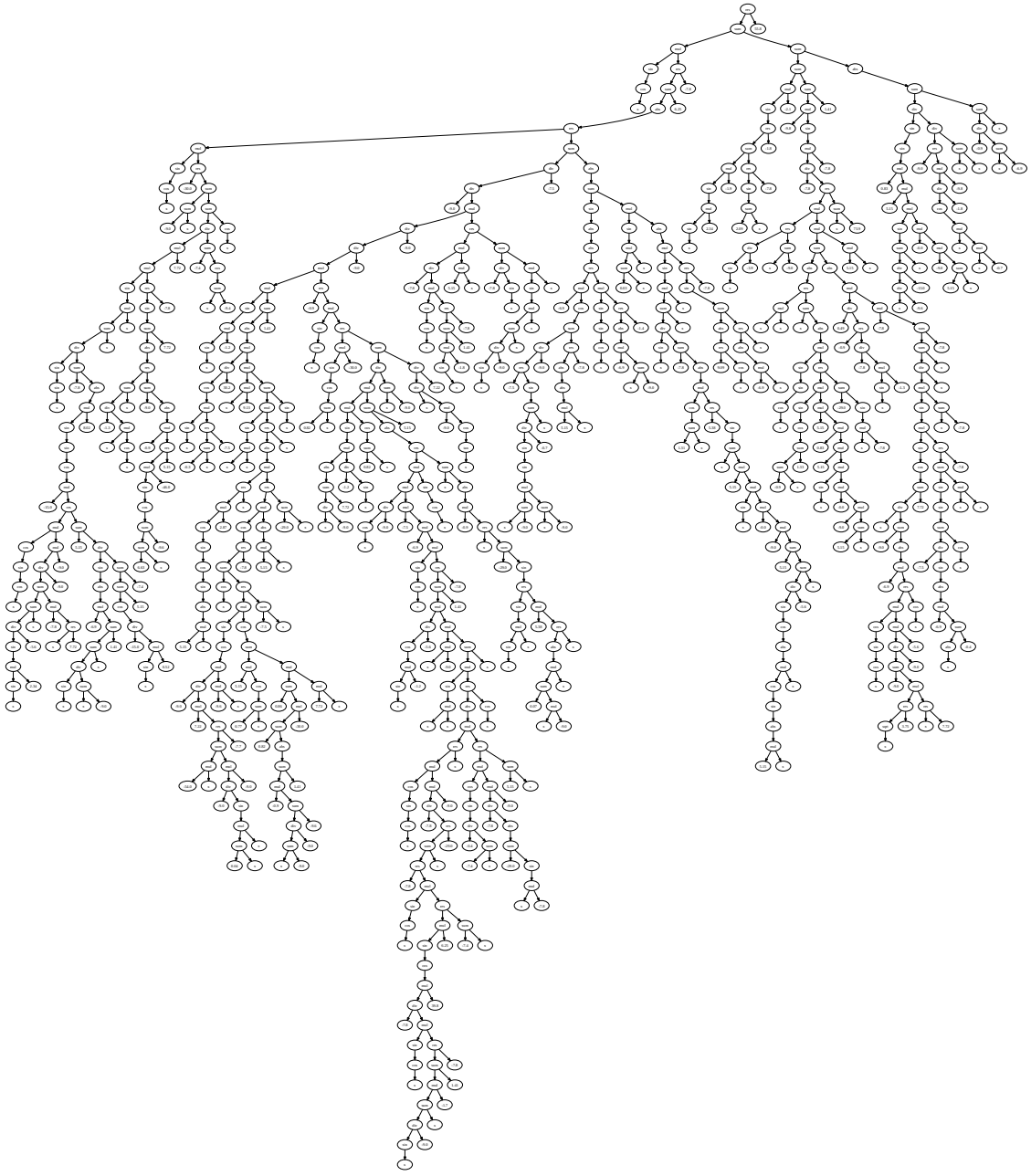


Figura 4.10: Árbol generado por IGP para la señal 1 con 844 nodos

Figura 4.11: Árbol generado por IGP para la señal 1 con 1648 nodos

Ahora se muestra la Tabla 4.3, donde se muestran las *aptitudes* correspondientes a las Figuras 4.8, 4.9, 4.10, 4.11 donde se puede observar relación entre el número de nodos con la *aptitud* obtenida.

Tabla 4.3: Comparación de Aptitudes entre *Eureqa* e IGP utilizando la señal 1

Sistema	Nodos	Aptitud
Eureqa	39	−2.38
IGP (Tam_Max=200)	191	−4.87
IGP (Tam_Max=1000)	699	−1.12
IGP (Tam_Max=2000)	1648	−1.04

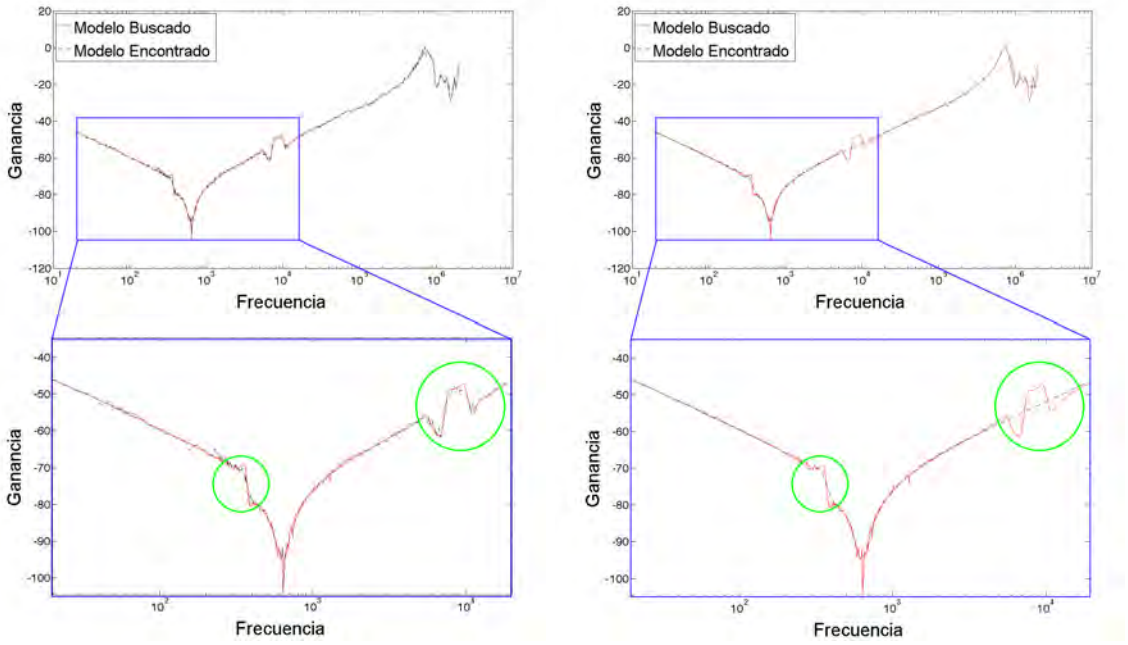
4.2.2. Señal 2

A continuación, se muestran los modelos obtenidos con IGP y *Eureqa* para la señal 2. Como se puede observar en la Figura 4.12 el comportamiento buscado está señalado con una linea continua color rojo, mientras que el comportamiento encontrado esta marcado con una linea punteada color negro. Este resultado nos demuestra que nuevamente el IGP ha obtenido un buen modelo con una *aptitud* de -0.66 a pesar de ser un comportamiento diferente al del sistema anterior, mientras que *Eureqa* logra obtener el modelo con una *aptitud* de -2.77, además se continúa observando que las lineas generadas por *Eureqa* son más suaves que las generadas por IGP, lo cuál provoca menor ajuste al comportamiento buscado.

En la Figura 4.13 se observa un acercamiento a la parte final de la señal 2, mostrando las zonas con esquinas, donde IGP se apega mejor al comportamiento buscado.

4.2.3. Señal 3

Ahora se presenta el resultado generado por IGP para la señal 3, el cual se muestra en la Figura 4.14, donde los colores para marcar los comportamientos, tanto encontrado como buscado son los mismos a los anteriores. Como se puede observar en este resultado, el modelo encontrado es un poco aceptable con una *aptitud* de -2.35, debido a que le ha costando mayor dificultad apegarse al comportamiento buscado, generando un comportamiento que se aleja del comportamiento buscado, sobre todo en la



(a) Modelo obtenido por IGP, *aptitud*= -0.66. (b) Modelo obtenido por *Eureqa*, *aptitud*= -2.77.

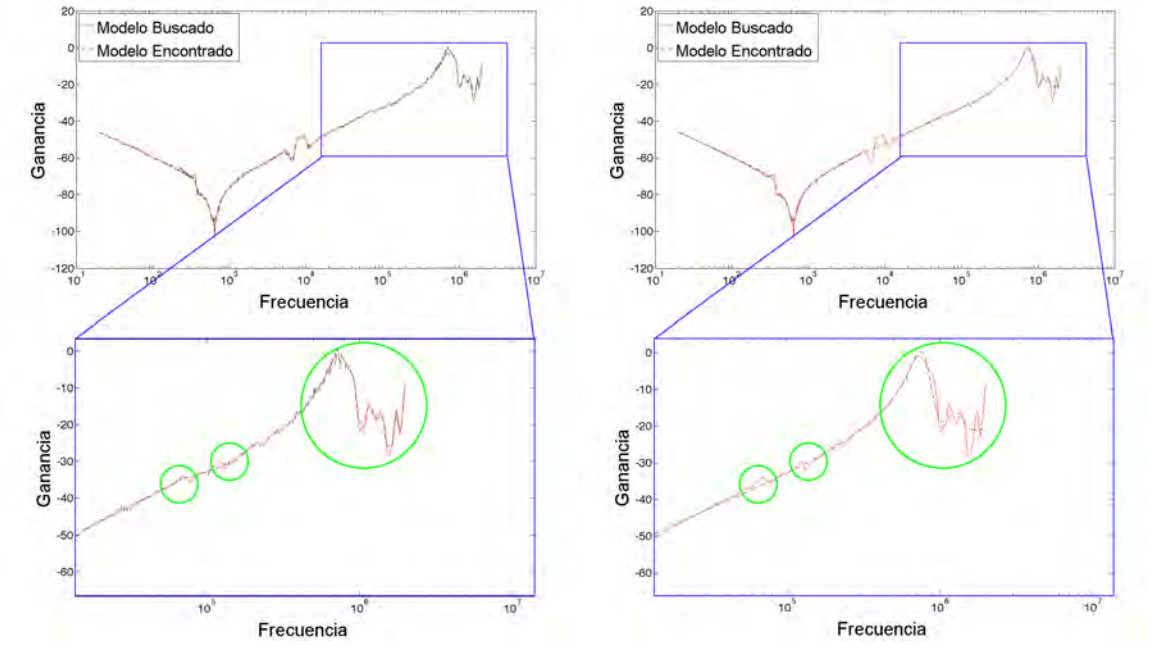
Figura 4.12: Mejores modelos obtenidos para la señal 2.

parte final del acercamiento a esta señal, debido a la complejidad que debe tener el modelo encontrado. En cuanto a *Eureqa*, su modelo encontrado es mucho mas alejado de lo que se buscaba encontrar, obteniendo una *aptitud* de -9.54.

En la Figura 4.15 se muestra un acercamiento a la parte final de la señal 3, mostrando nuevamente un comportamiento muy lejano al esperado por parte de *Eureqa*, mientras que con IGP el comportamiento es mejor.

4.2.4. Señal 4

Enseguida se muestra el resultado obtenido por IGP para la señal 4, donde inicialmente se pensaba que se tendrían malos resultados, ya que visualmente se considera con un mayor grado de dificultad por tantas puntas que suben y bajan, pero los resultados arrojados fueron muy satisfactorios, obteniendo un buen modelo usando IGP con una *aptitud* de -1.47, como se puede observar en la Figura 4.16 en el acercamiento a la primer parte de esta señal, el ajuste es mucho mejor al ajuste obtenido por



(a) Modelo obtenido por IGP, *aptitud*= -0.66 (b) Modelo obtenido por *Eureqa*, *aptitud*= -2.77

Figura 4.13: Mejores modelos obtenidos para la señal 2.

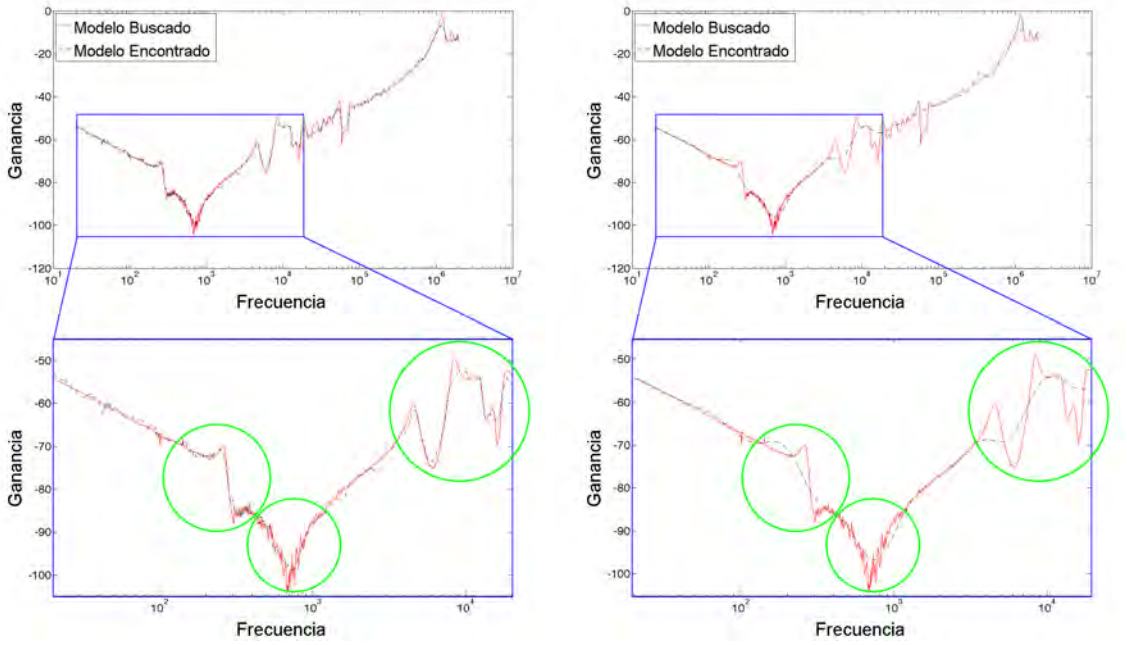
Eureqa.

En la Figura 4.17 se realiza un acercamiento a la segunda mitad de la señal 4, observando una zona con gran cantidad de esquinas y observando el buen ajuste que obtiene IGP, mientras que *Eureqa* genera una linea curva muy suave, provocando que su *aptitud* sea de -9.78.

4.2.5. Señal 5

A continuación se muestra el modelo obtenido por IGP para la señal 5. Como se puede ver en la Figura 4.18 este sistema es muy parecido al anterior, pero no son los mismos. En esta figura se muestra que el modelo encontrado se apega bien al comportamiento del sistema real con una *aptitud* de -1.83, mientras que *Eureqa* logra obtener una *aptitud* de -9.79. Donde nuevamente se logra ver las diferencias tanto en *aptitud* como en comportamiento.

En la Figura 4.19 se observa un acercamiento a la parte final de la señal 5, donde



(a) Modelo obtenido por IGP, *aptitud*= -2.35. (b) Modelo obtenido por *Eureqa*, *aptitud*= -9.54.

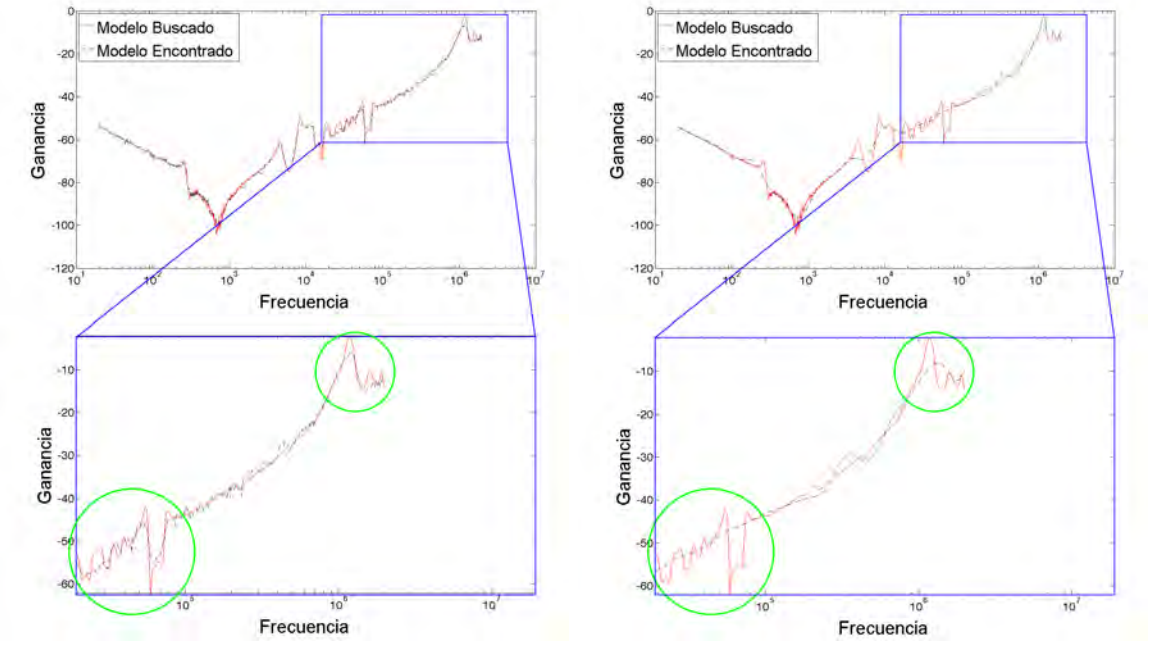
Figura 4.14: Mejores modelos obtenidos para la señal 3.

de nueva ocasión en zonas con varias esquinas el comportamiento obtenido por *Eureqa* no es lo que se esperaba.

4.2.6. Señal 6

En la Figura 4.20 se muestra el modelo encontrado para la señal 6, que como se observa la parte inicial del resultado obtenido por IGP, el comportamiento no se apega al comportamiento buscado, pero en la parte central y final se ajusta bien. Este modelo tiene un *aptitud* de -8.71, mientras que *Eureqa* logra obtener una *aptitud* de -13.47, generando para esta parte de la señal, una line muy suave.

En la Figura 4.21 se puede observar un acercamiento a la parte final de la señal 6, donde el comportamiento encontrado por *Eureqa* para esta parte de la señal sigue siendo muy suave, evitando un buen ajuste, mientras que IGP logra obtener un mejor resultado.



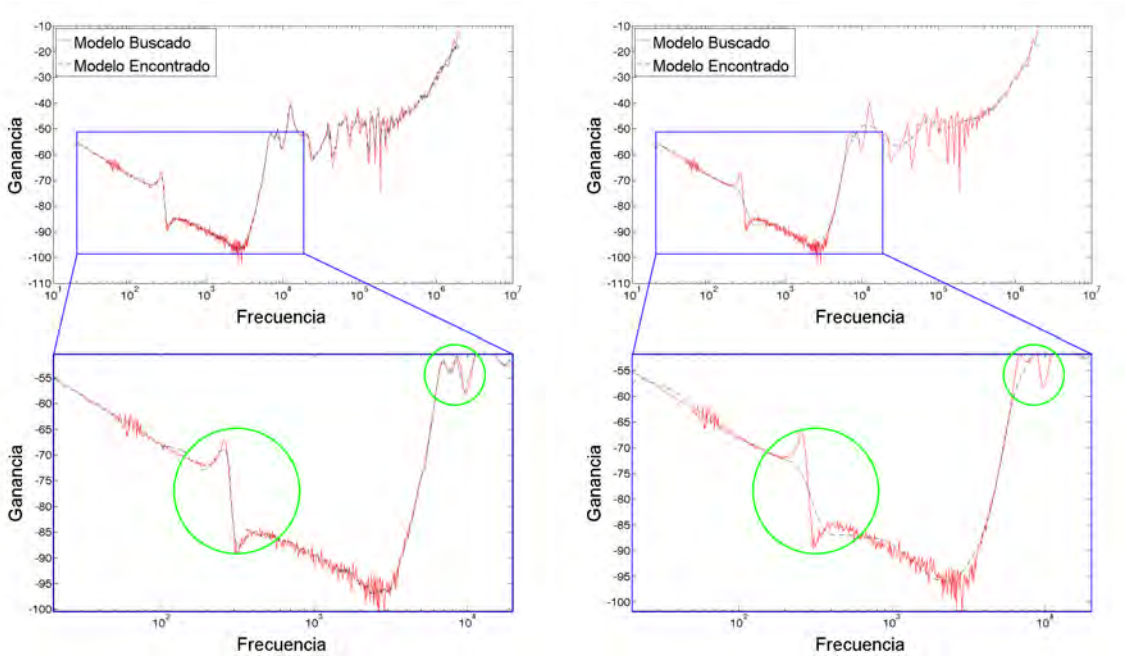
(a) Modelo obtenido por IGP, *aptitud*= -2.35. (b) Modelo obtenido por *Eureqa*, *aptitud*= -9.54.

Figura 4.15: Mejores modelos obtenidos para la señal 3.

4.2.7. Señal 7

La Figura 4.22 muestra una señal muy distinta a las anteriores, de la cual se pensaba que el modelo encontrado se ajustaría sin problemas mejor que los modelos anteriores, donde a pesar de que parece tener un grado de dificultad menor se pueden observar zonas en donde el comportamiento encontrado por IGP no se apega tanto como se esperaba, obteniendo una *aptitud* de -0.78, lo cual perecería que se debe ajustar mejor al ver este valor, pero no es así y esto de debe a que existen muchas zonas donde el ajuste es bueno ayudando a que el valor de dicho *aptitud* sea pequeño. Mientras tanto, en esta parte de la señal *Eureqa* no tuvo problemas para lograr un buen ajuste, aunque su *aptitud* fue de -2.30.

En la Figura 4.23 se logra ver un acercamiento a la parte final de la señal 7, observando el motivo por que IGP logra obtener mejor *aptitud* debido a que en esta parte el ajuste de IGP es mejor, aunque con complicaciones en varias partes de la señal.



(a) Modelo obtenido por IGP, *aptitud*= -1.47 (b) Modelo obtenido por *Eureqa*, *aptitud*= -9.78.

Figura 4.16: Mejores modelos obtenidos para la señal 4.

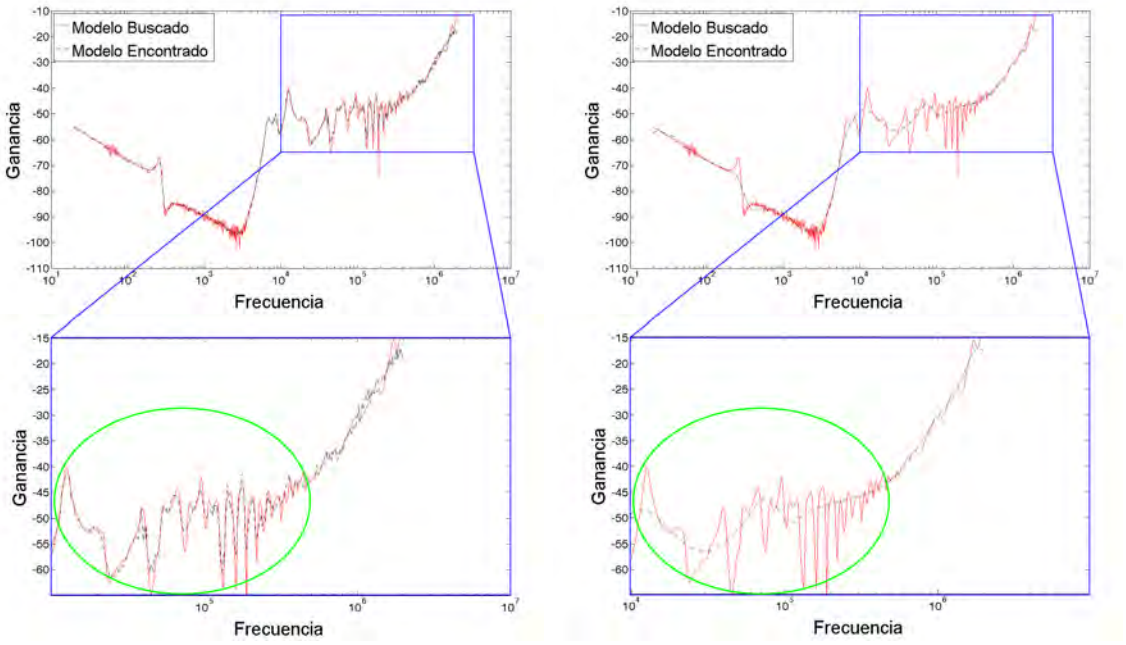
4.2.8. Señal 8

En la Figura 4.24 se muestra el modelo encontrado para la señal 8 la cual nuevamente no tiene gran parecido a las anteriores, teniendo varias curvas en su parte central. A pesar de tal diferencia IGP continúa encontrando un buen modelo para esta señal, presentando un modelo con *aptitud* de -1.03, mientras que *Eureqa* presenta un modelo con *aptitud* de -8.31, observando mejor ajuste por parte de IGP.

En la Figura 4.25 se muestra un acercamiento a la parte final de la señal 8, observando que tanto a IGP como *Eureqa* se les ha complicado apegarse en esta zona, al comportamiento deseado.

4.2.9. Señal 9

La Figura 4.26 muestra la señal 9, que es muy similar a la señal 8 de la Figura , observando que este sistema nuevamente presenta varias esquinas en su parte central,



(a) Modelo obtenido por IGP, *aptitud*= -1.47. (b) Modelo obtenido por *Eureqa*, *aptitud*= -9.78.

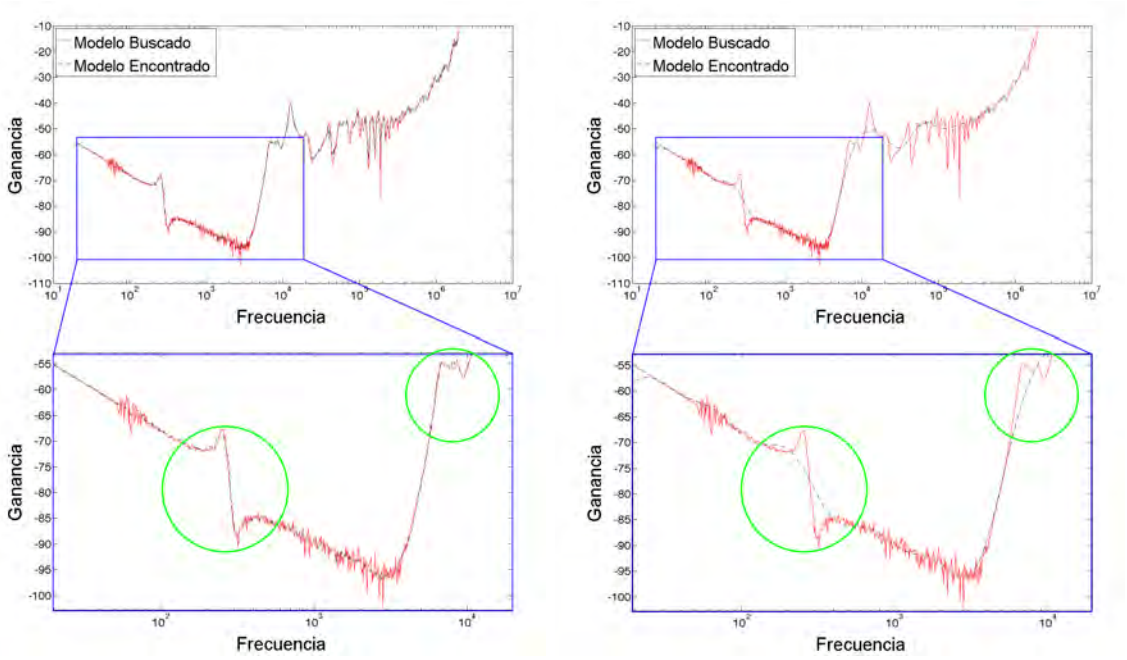
Figura 4.17: Mejores modelos obtenidos para la señal 4.

y a pesar de eso, el modelo encontrado por IGP es muy similar al modelo real, presentando una *aptitud* de -1.13. *Eureqa* ha logrado obtener un modelo con una *aptitud* de -9.42, lo cual es una diferencia notable debido nuevamente a su ajuste al comportamiento buscado.

En la Figura 4.27 se muestra un acercamiento a la parte final de la señal 9, observando que tanto IGP como *Eureqa* no logran ajustarse bien a esta parte de la señal 9.

4.3. conclusión

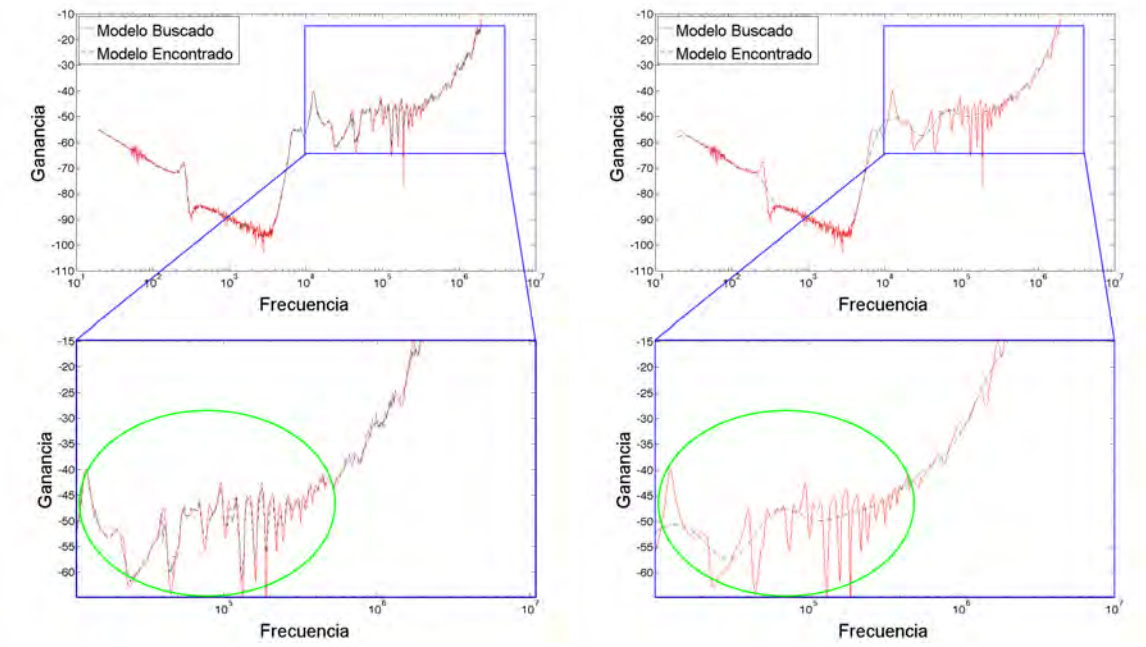
Este capítulo a mostrado los resultados de los mejores modelos encontrados por IGP y *Eureqa* para las nueve diferentes señales de transformadores de potencia, observando que en todos los casos, los resultados obtenidos por IGP han logrado superar a los obtenidos por *Eureqa*. También se pudo observar que PG sin hacer uso de la



(a) Modelo obtenido por IGP, *aptitud*= -1.83. (b) Modelo obtenido por *Eureqa*, *aptitud*= -9.79.

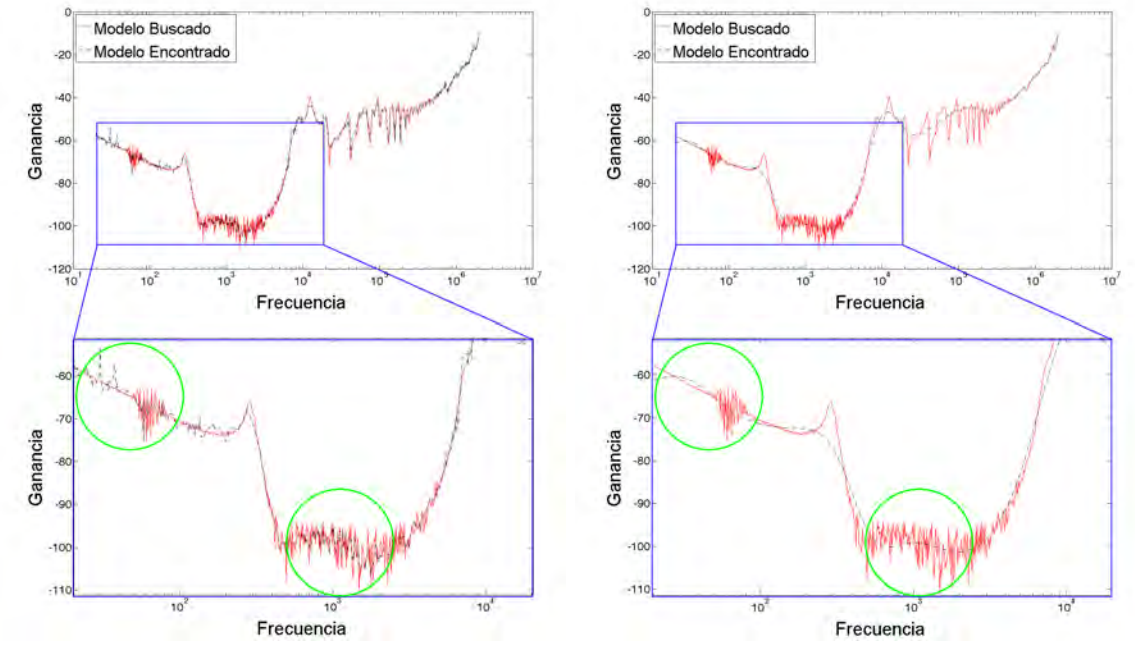
Figura 4.18: Mejores modelos obtenidos para la señal 5.

estrategia propuesta no ha logrado superar a los resultados obtenidos por IGP y esto se puede ver en las Tablas 4.1 y 4.2, mientras que en la Tabla 4.3 se pudo observar que *Eureqa* no logra obtener resultados tan aceptables, pero el número de nodos es muy bajo comparado con los mejores resultados obtenidos por IGP.



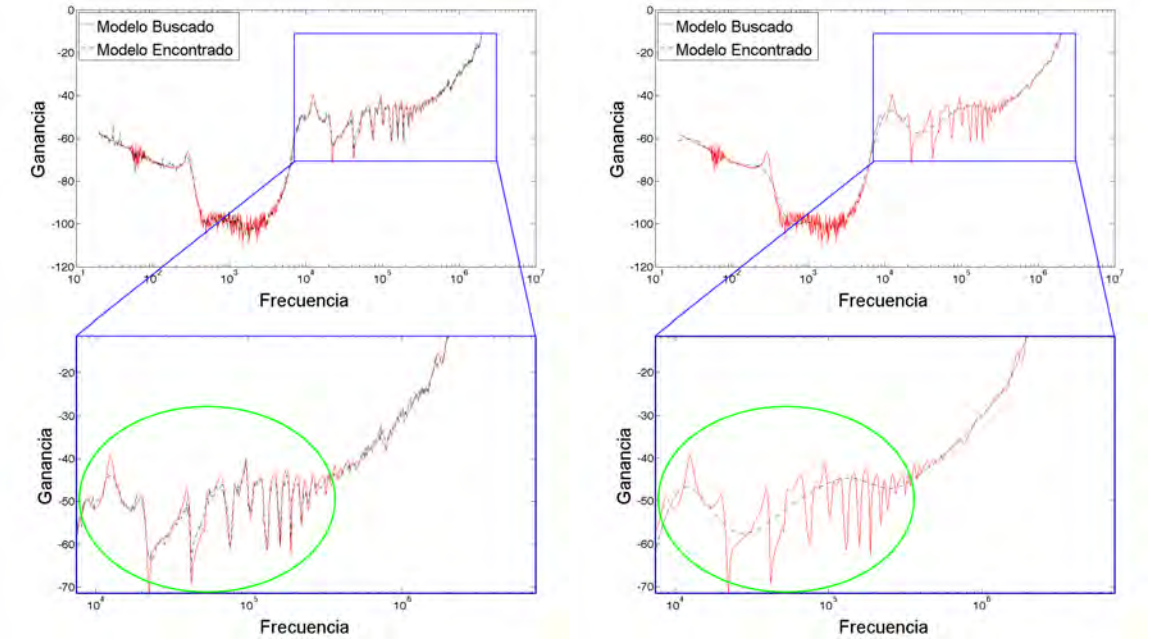
(a) Modelo obtenido por IGP, *aptitud*= -1.83. (b) Modelo obtenido por *Eureqa*, *aptitud*= -9.79.

Figura 4.19: Mejores modelos obtenidos para la señal 5.



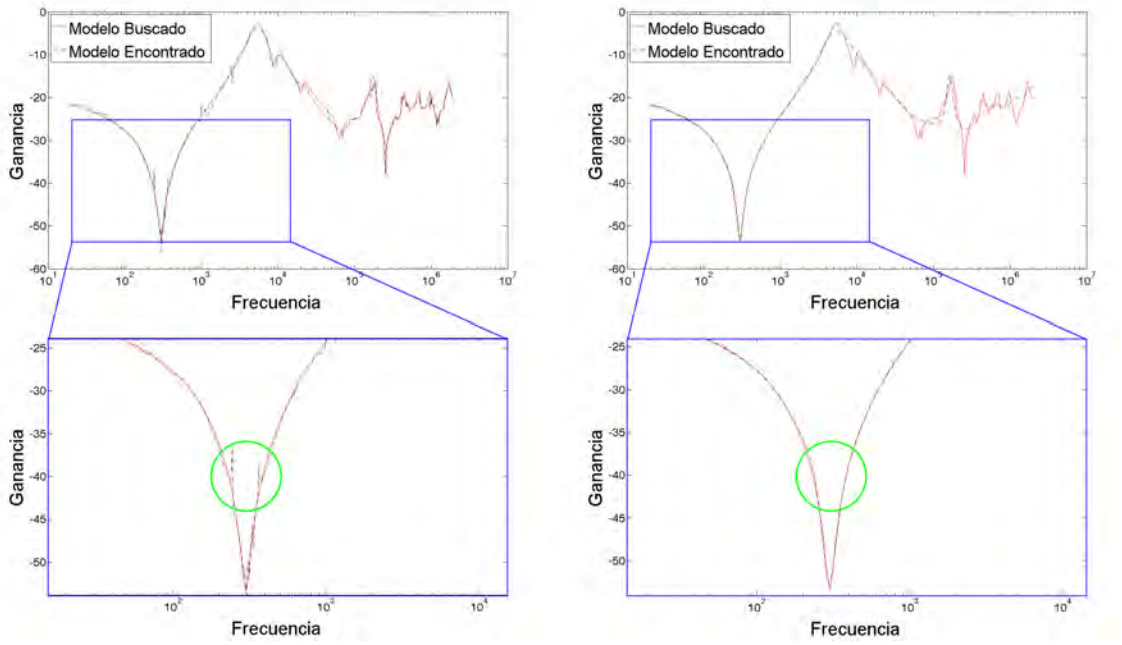
(a) Modelo obtenido por IGP, *aptitud*= -8.71. (b) Modelo obtenido por *Eureqa*, *aptitud*= -13.47.

Figura 4.20: Mejores modelos obtenidos para la señal 6.



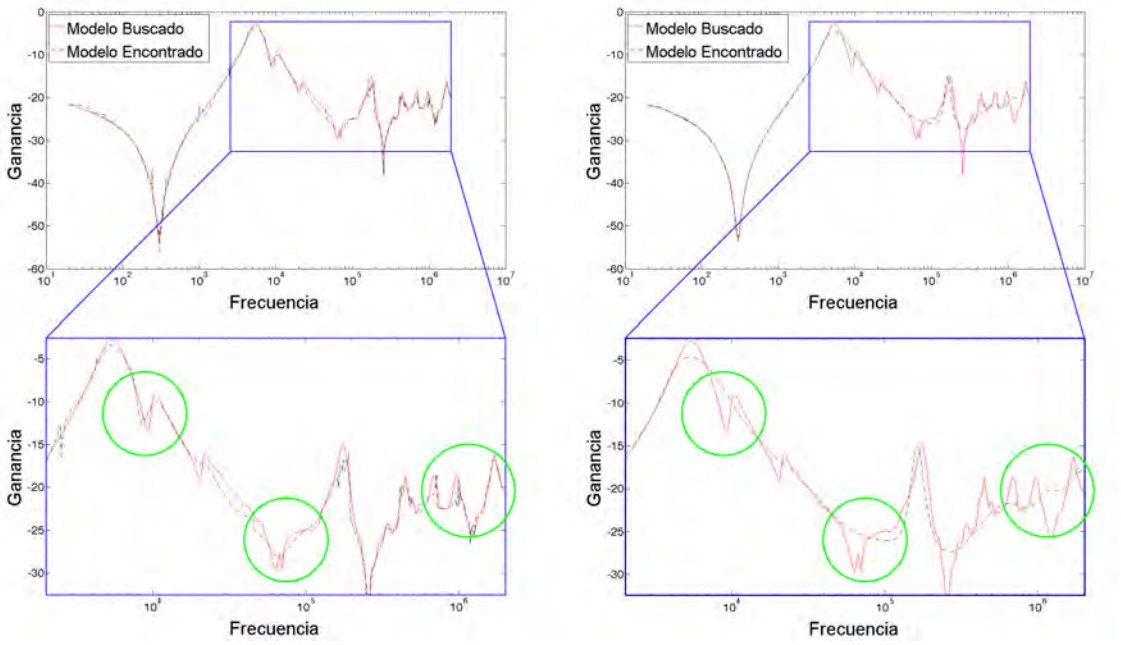
(a) Modelo obtenido por IGP, *aptitud*= -8.71 (b) Modelo obtenido por *Eureqa*, *aptitud*= -13.47.

Figura 4.21: Mejores modelos obtenidos para la señal 6.



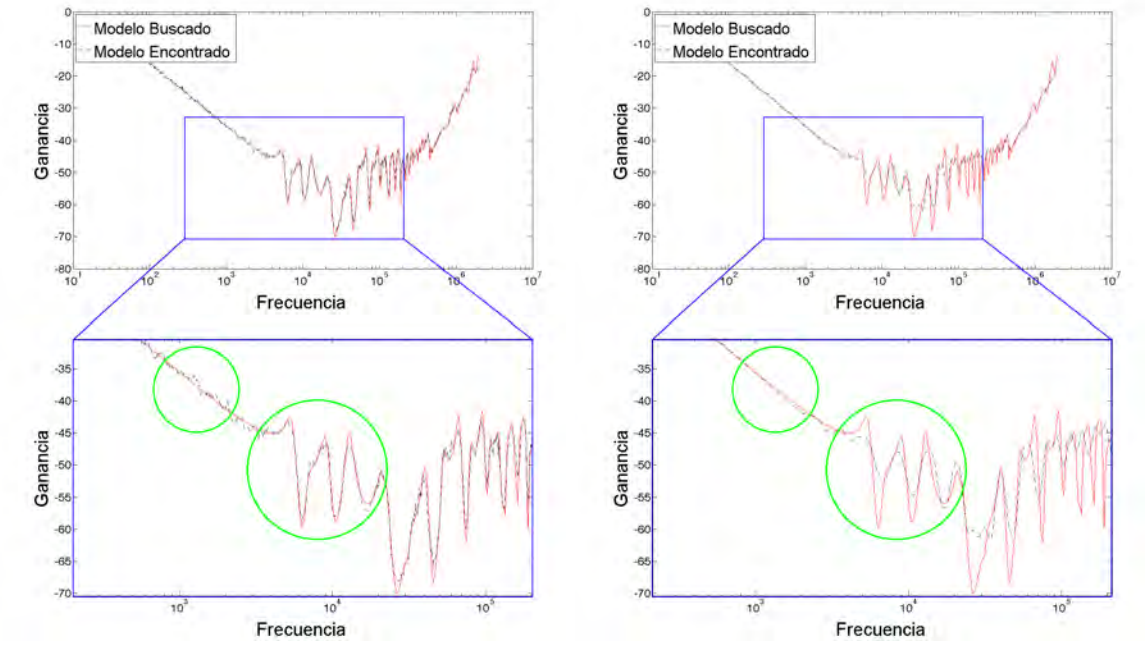
(a) Modelo obtenido por IGP, *aptitud*= -0.78. (b) Modelo obtenido por *Eureqa*, *aptitud*= -2.30.

Figura 4.22: Mejores modelos obtenidos para la señal 7.



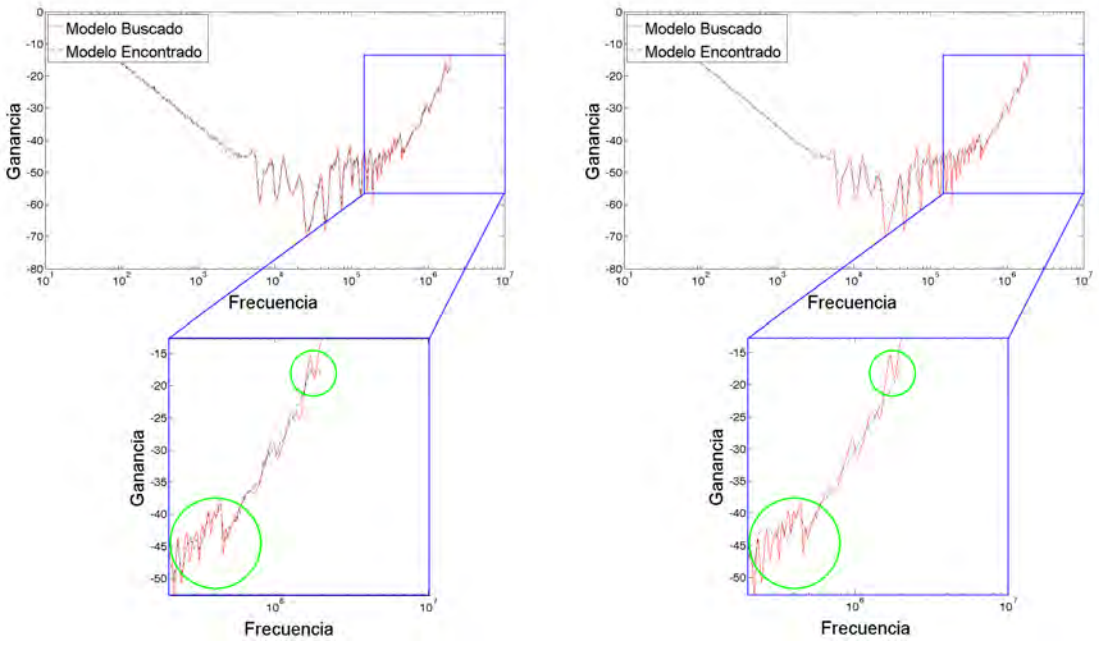
(a) Modelo obtenido por IGP, *aptitud*= -0.78. (b) Modelo obtenido por *Eureqa*, *aptitud*= -2.30.

Figura 4.23: Mejores modelos obtenidos para la señal 7.



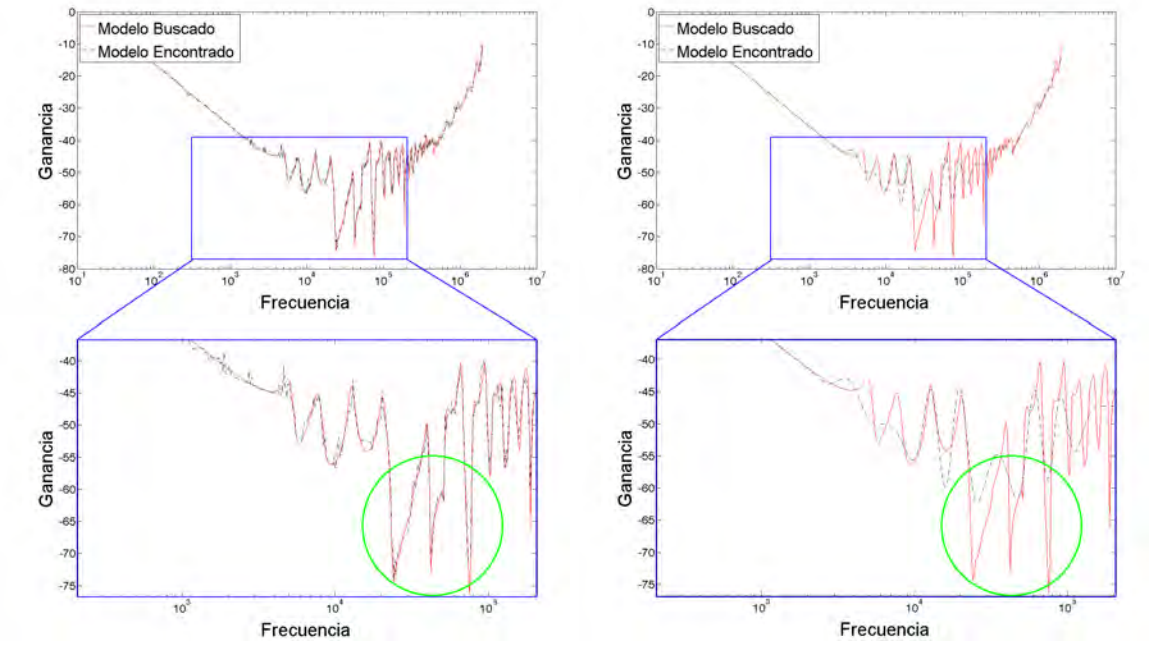
(a) Modelo obtenido por IGP, *aptitud*= -1.03. (b) Modelo obtenido por *Eureqa*, *aptitud*= -8.31.

Figura 4.24: Mejores modelos obtenidos para la señal 8.



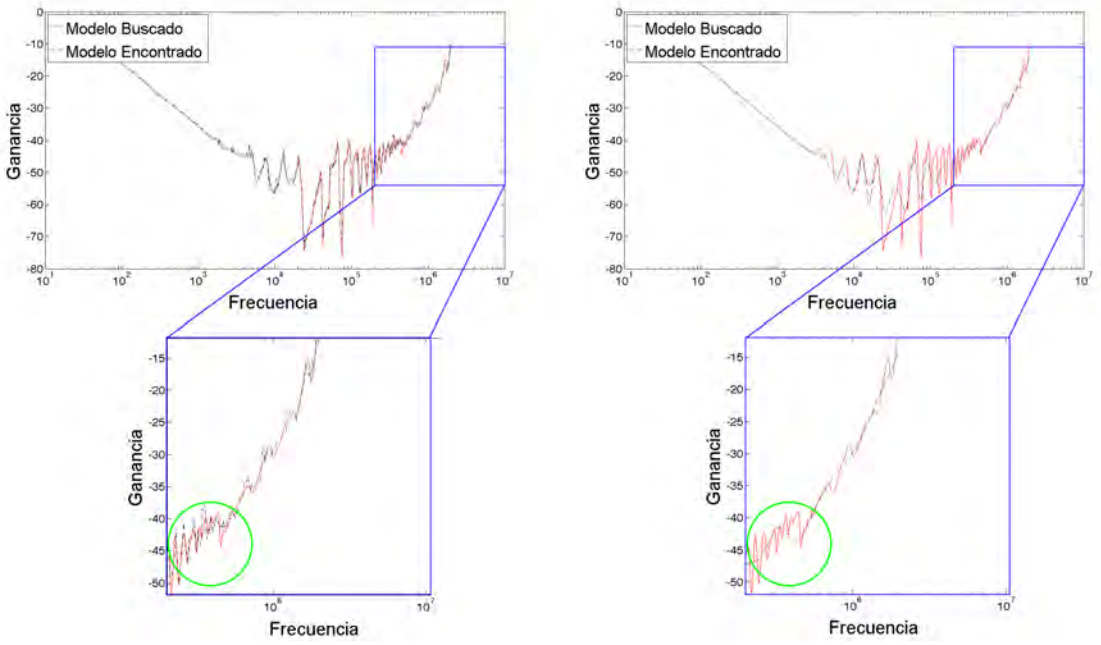
(a) Modelo obtenido por IGP, *aptitud*= -1.03. (b) Modelo obtenido por *Eureqa*, *aptitud*= -8.31.

Figura 4.25: Mejores modelos obtenidos para la señal 8.



(a) Modelo obtenido por IGP, *aptitud*= -1.13. (b) Modelo obtenido por *Eureqa*, *aptitud*= -9.42.

Figura 4.26: Mejores modelos obtenidos para la señal 9.



(a) Modelo obtenido por IGP, *aptitud*= -1.13. (b) Modelo obtenido por *Eureqa*, *aptitud*= -9.42.

Figura 4.27: Mejores modelos obtenidos para la señal 9.

Capítulo 5

Conclusiones

En este capítulo se plasman las conclusiones obtenidas con el desarrollo de la presente tesis, así como una breve reseña de los pendientes que se pretenden llevar a cabo en un futuro.

Con los resultados obtenidos por el sistema IGP se demuestra nuevamente la capacidad de la Programación Genética para la identificación de sistemas y se muestra que la estrategia propuesta logra obtener modelos muy cercanos al comportamiento de los sistemas con los que se experimentó, los cuales son comportamientos de transformadores de potencia. Además cabe resaltar que se ha experimentado con problemas reales y de importancia industrial, lo que no es muy común observar en esta área.

Como comprobación de que el sistema desarrollado funciona, se experimentó con un total de nueve señales diferentes, para la cuales se han encontrado modelos con comportamientos similares al real. Además se compararon los resultados obtenidos con el sistema *Eureqa*, donde los resultados más precisos fueron obtenidos por IGP.

Hasta ahora, los modelos encontrados por IGP tiene un gran tamaño en nodos, es decir, símbolos que pueden ser operadores aritméticos, trigonométricos, variables o constantes. Mientras que los modelos obtenidos con *Eureqa* son mucho más pequeños, por lo que se dedujo que de alguna forma esta herramienta de software le da prioridad a sus modelos con menor tamaño, o se centra en generarlos de un tamaño pequeño provocando que esto afecte la evolución de sus modelos cuando el comportamiento a buscar es muy complejo. Por ahora IGP no se está centrando en obtener modelos

tan pequeños, pero los resultados obtenidos son notablemente más precisos que los obtenidos por *Eureqa*.

5.1. Trabajo Futuro

Debido a la estructura de los modelos encontrados y el tiempo que tarda el sistema en encontrarlos, se propone el siguiente trabajo a futuro, teniendo como objetivo mejorar el rendimiento de dicho sistema:

- Reducir modelos encontrados. Esto con el fin de simplificar los modelos para ser más comprensibles, tomando en cuenta estrategias del área del álgebra simbólica que pueden ser aplicadas en la reducción de los árboles, ya que su uso está fuera del alcance de este trabajo.

- Optimizar los valores de las terminales mediante alguna técnica como *Backpropagation* para el mejoramiento de los modelos encontrados.

- Paralelizar IGP. La estructura del sistema al igual que la forma en que trabaja PG, por ejemplo la evaluación de los individuos de la población, se presta para aprovechar las nuevas tecnologías de los equipos de cómputo, como lo es la manera de realizar varios procesos a la vez haciendo uso de la programación en paralelo, reduciendo notablemente el tiempo para encontrar los modelos.

Referencias

[Allen, 1971] Allen, D. M. (1971). Mean square error of prediction as a criterion for selecting variables. *Technometrics*, 13(3):469–475.

[Allgaier and McDevitt, 2013] Allgaier, N. and McDevitt, R. (2013). Reverse engineering the brain with eureqa. (0):7. PMID: 23596539.

[Augusto and Barbosa, 2000] Augusto, D. and Barbosa, H. J. C. (2000). Symbolic regression via genetic programming. In *Sixth Brazilian Symposium on Neural Networks, 2000. Proceedings*, pages 173–178.

[Avalos, 2008] Avalos, A. (2008). Field and laboratory experiences using sweep frequency response analysis in power transformers. *Doble Engineering Company -75th Annual International Doble Client Conference*.

[Blickle and Thiele, 1995] Blickle, T. and Thiele, L. (1995). A comparison of selection schemes used in genetic algorithms.

[Dubčáková, 2011] Dubčáková, R. (2011). Eureqa: software review. *Genetic Programming and Evolvable Machines*, 12(2):173–178.

[Flores and Graff, 2005] Flores, J. J. and Graff, M. (2005). System identification using genetic programming and gene expression programming. In Yolum, p., Güngör, T., Gürgen, F., and Özturan, C., editors, *Computer and Information Sciences - ISCIS 2005*, number 3733 in Lecture Notes in Computer Science, pages 503–511. Springer Berlin Heidelberg.

[Flores and Graff, 2006] Flores, J. J. and Graff, M. (2006). Lessons learned in modeling dynamic systems using genetic programming. *Research in Computing Science*, 26:177–186.

[Gagné and Parizeau, 2006] Gagné, C. and Parizeau, M. (2006). Open beagle a c++ framework for your favorite evolutionary algorithm. *SIGEVOlution*, 1(1):12,15.

[Giacobini et al., 2002] Giacobini, M., Tomassini, M., and Vanneschi, L. (2002). Limiting the number of fitness cases in genetic programming using statistics. In Guervós, J. J. M., Adamidis, P., Beyer, H.-G., Schwefel, H.-P., and Fernández-Villacañas, J.-L., editors, *Parallel Problem Solving from Nature — PPSN VII*, number 2439 in Lecture Notes in Computer Science. Springer Berlin Heidelberg.

[Graff et al., 2013] Graff, M., Pena, R., and Medina, A. (2013). Wind speed forecasting using genetic programming. In *2013 IEEE Congress on Evolutionary Computation (CEC)*, pages 408–415.

[Holland, 1992] Holland, J. H. (1992). *Adaptation in Natural and Artificial Systems: An Introductory Analysis With Applications to Biology, Control, and Artificial Intelligence*. MIT Press. (First edition 1975, Ann Arbor: University of Michigan Press.).

[Keim, 2009a] Keim, B. (2009a). Computer program self-discovers laws of physics. *Wired*.

[Keim, 2009b] Keim, B. (2009b). Download your own robot scientist. *Wired*.

[Koza, 1992] Koza, J. R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press.

[Luke and Spector, 1997] Luke, S. and Spector, L. (1997). A comparison of crossover and mutation in genetic programming. In *Genetic Programming 1997: Proceedings of the Second Annual Conference*, page 240–248. Morgan Kaufmann.

[Luke and Spector, 1998] Luke, S. and Spector, L. (1998). A revised comparison of crossover and mutation in genetic programming. In *Genetic Programming 1997: Proceedings of the Second Annual Conference*, page 240–248. Morgan Kaufmann.

[Poli et al., 2008] Poli, R., Langdon, W. B., and McPhee, N. F. (2008). *A Field Guide to Genetic Programming*. Lulu Enterprises, UK Ltd.

[Schmidt M., 2009] Schmidt M., L. H. (2009). Distilling free-form natural laws from experimental data. *Science*, 324(5923):81–85.

[Spears et al., 1992] Spears, W. M., Pears, E.-m., and Mil, A. N. N. (1992). Crossover or mutation? In *Foundations of Genetic Algorithms 2*, page 221–237. Morgan Kaufmann.

[Srečec et al., 2013] Srečec, S., Ceh, B., Ciler, T. S., and Rus, A. F. (2013). Empiric mathematical model for predicting the content of alpha-acids in hop (humulus lupulus l.) cv. aurora. *SpringerPlus*, 2(1):59. PMID: 23596559.

[Thierens and Goldberg, 1994] Thierens, D. and Goldberg, D. (1994). Convergence models of genetic algorithm selection schemes. In Davidor, Y., Schwefel, H.-P., and Männer, R., editors, *Parallel Problem Solving from Nature — PPSN III*, number 866 in Lecture Notes in Computer Science, pages 119–129. Springer Berlin Heidelberg.

[White, 2012] White, D. R. (2012). Software review: the ECJ toolkit. *Genetic Programming and Evolvable Machines*, 13(1):65–67.

[White and Poulding, 2009] White, D. R. and Poulding, S. (2009). A rigorous evaluation of crossover and mutation in genetic programming. In Vanneschi, L., Gustafson, S., Moraglio, A., Falco, I. D., and Ebner, M., editors, *Genetic Programming*, number 5481 in Lecture Notes in Computer Science, pages 220–231. Springer Berlin Heidelberg.

[Whitley, 1989] Whitley, D. (1989). The GENITOR algorithm and selection pressure: Why rank-based allocation of reproductive trials is best. In *Proceedings of the Third*

International Conference on Genetic Algorithms, page 116–121. Morgan Kaufmann Publishers Inc.

[Wilson et al., 2004] Wilson, G. C., Intyre, A. M., and Heywood, M. I. (2004). Resource review: Three open source systems for evolving Programs,ÄìLilgp, ECJ and grammatical evolution. *Genetic Programming and Evolvable Machines*, 5(1):103–105.