Una Aproximación a la Programación Genética Aplicada a Problemas de Programación Inductiva

Edwin Sneyder Gantiva Ramos[1]

Universidad Nacional de Colombia, Bogotá, Colombia,
I.esgantivar@unal.edu.co

Abstract. La computación evolutiva posee diferentes campos de estudio entre estos se encuentran los algoritmos genéticos, estos se basan en conceptos de evolución natural inspirados en la teoría Darwiniana, estos algoritmos se basan en la construcción de un cromosoma que va ser la representación del problema a optimizar. Conociendo esto podremos introducir el problema de la programación genética, está en términos simples es una especialización de los algoritmos genéticos donde la representación genotípica de su cromosoma será un árbol sintáctico, este aparente sencillo cambio implica profundos cambios en lo que compete a el desarrollo de algoritmos genéticos. Este documento tiene como propósito hacer una aunque breve, ambiciosa aproximación al problema de la programación genética.

Keywords: Optimización, Algoritmos Geneticos, Operadores Geneticos Auto Adapativos, Programación Genetica,

1 Introducción

Comúnmente la comunidad interesada en los temas de la computación evolutiva codifican sus problemas en representaciones genotipicas como números binarios ó reales. Esto finalmente termina reduciendo considerablemente la complejidad en la construcción de los operadores genéticos que se verán involucrados en los procesos evolutivos. En este punto es donde la programación genética representa un reto adicional, puesto que la representación de este problema se hace por medio de arboles sintácticos y es bien conocido que estas estructuras tienen características muy especiales y fuertemente condicionadas. Por este motivo la construcción de operadores genéticos sera de especial atención y cuidado.

Este documento esta estructurado de la siguiente manera, tendrá una sección destinada a mostrar las generalidades del lenguaje de programación FUNICO [1] que fue desarrollado por el Profesor Camilo Cubides, y que es sumamente importante para poder dar desarrollo a este trabajo, dado que este se basa en el interprete y el lenguaje FUNICO. La siguiente sección hará una breve descripción de la teoría que se encuentra detrás de este trabajo, posterior a esto

se expondrán las ideas que se han propuesto para dar desarrollo a este trabajo y terminando con una sección donde se mostraran los resultados conseguidos.

2 Estado del Arte

En esta sección se hará una breve descripción del lenguaje FUNICO, conceptos necesarios para poder tener un entendimiento del trabajo desarrollado.

FUNICO tiene como base 5 constantes y estas son:

- 0 Que representa el valor 0, y el cardinal del conjunto vació.
- *true* Que representa el valor lógico verdadero.
- $false$ Que representa el valor lógico falso.
- [−] Representa la lista vacía.
- *undef* Representa el símbolo indefinido, se usa para definir una expresión que no puede computarse.

De las anteriores constantes, para el desarrollo del trabajo solo tendremos en cuenta las primeras 3.

FUNICO posee las siguientes funciones primitivas:

- $s(\_)$ Permite calcular la función sucesor.
- $equal(\_, \_)$ Predicado que define que dos expresiones son iguales.
- $[\_|\_]$ Es el constructor básico de listas.

De las funciones primitivas, en el desarrollo de este trabajo se hace uso de las dos primeras.

Finalmente dentro de FUNICO las variables deben ser definidas en mayúsculas.

3 Marco Teórico

3.1 Definición

La Programación Genética es un campo de estudio de los algoritmos genéticos, en donde los individuos (cromosomas) de la población son programas de computador [3].

El objetivo de la programación genética es encontrar un programa de computador que de solución a un problema en donde encontrar una solución analítica sea un trabajo difícil.

Fitness: La función de fitness de un individuo esta basada en la cobertura sobre los ejemplos presentes en el conjunto de entrenamiento.

Los programas pueden ser representados por un árbol sintáctico o un conjunto de arboles sintáctico, los cromosomas son generalmente representados con una estructura de árbol computacional. De esta estructura se extraerá una función en el formato de FUNICO y se hará uso del interprete para evaluar el cromosoma.

3.2 Algoritmo Genético Convencional

El algoritmo genético convencional es el concepto base para el desarrollo de este documento, este en términos sencillos busca dada una función de fitness y un conjunto de población inicial busca evolucionar el rendimiento del conjunto de los individuos sobre la función de fitness generación tras generación.

Algorithm 3.1: Algoritmo Genetico($f, \mu, CondicionDeTerminacion$)

```
1 Inicializar la Población Inicial P_0;
2 Evaluar(P_0, f);
3 while CondiciondeTerminacion do
4 |   descendientes = GENERACION(P_t, f, SELECCION);
5 |   P_t = FUNCION_REEMPLAZO(descendientes, P_{t-1})
```

.

Para el desarrollo de este proyecto se hará uso de operadores genéticos auto adaptativos que fueron presentados en [2]. Conceptos que han sido implementados en trabajos personales previos a este.

4 Propuesta

4.1 Población Inicial

La población inicial ha sido generada por medio de un espacio generador de ecuaciones, funciones y terminales. Teniendo como criterio de parada un limite de profundidad del árbol sintáctico. Sin embargo este método de generación fue deficiente en el proceso de generación de poblaciones iniciales. Fue necesario dar mas probabilidad para que funciones que tienen un retorno booleano, las terminales booleanas tengan mas peso que las terminales normales.

4.2 Criterio de Selección

El criterio de selección usado es el tipo ruleta, que se basa en ofrecer "tickets" de la ruleta acorde al fitness, esto quiere decir que todos los individuos por mas

deficiente que sea su valor de fitness tendrá una oportunidad de ser escogido, aunque obviamente un individuo con mejor fitness tendrá mas probabilidad de ser escogido.

4.3 Criterio de Reemplazo

Para el desarrollo de este documento se uso un criterio de reemplazo de estado estable, esto es que solo se permite que un descendiente reemplace a su pariente si su fitness es mejor o al menos igual, esto esta implementado en trabajos personales previos a este.

4.4 Operadores Genéticos

- **External Deep Swap**: Este operador tiene una ariedad de 2. Dados dos individuos, de manera aleatoria se elige una ecuación por cada individuo. Una vez se tienen las ecuaciones se elige un nodo de manera aleatoria de cada ecuación y estos son intercambiados.
- **Internal Swap**: Este operador tiene una ariedad de 1. Dado un individuo, se elige de manera aleatoria dos ecuaciones de su programa, y por cada ecuación se elige un nodo de manera aleatorio y estos son intercambiados.
- **Mutation Equation**: Este operador tiene una ariedad de 1. Dado un individuo, de manera aleatoria se escoge una función y esta es cambiada por una totalmente nueva que ha sido generada aleatoriamente, siguiendo el mismo criterio que se uso en la generación de la población inicial.
- **Mutation Internal**: Este operador tiene una ariedad de 1. Dado un individuo, de manera aleatoria se escoge una ecuación de el programa, sobre esta ecuación se elige un nodo de manera aleatoria, y es reemplazado por otro nodo generado de manera aleatoria.
- **Swap**: Este operador tiene una ariedad de 1. Dado un individuo, de manera aleatoria se eligen dos ecuaciones y se intercambia su orden interno.
- **XOverEquation**: Este operador tiene una ariedad 2. Dados dos individuos, de manera aleatoria se escoge una ecuación por cada individuo y se intercambian de individuo a individuo.

4.5 Función de Fitness

La función de fitness se basa en el porcentaje de cobertura del individuo (programa) sobre los ejemplos que se tienen dentro del conjunto de entrenamiento. Siendo así el menor valor de cobertura 0.0 y el mayor 1.0. Es importante mencionar que el hecho que un programa inducido tenga un valor de cobertura de 1.0 no implica que prediga todo el modelo, pues hay que tener presente el que se conoce como **No Free Lunch Theorem**.

Nota : El desarrollo que se ha planteado anteriormente se encuentra disponible para su uso y consulta en Repositorio en GitHub.

5 Resultados

Para los resultados del desarrollo de este trabajo se han desarrollado un algoritmo que necesita como parámetros de inicio, adicionales a los que son usuales dentro de los algoritmos genéticos. La lista de ejemplos positivos, la lista de funciones que intervienen en la función a inducir, un arreglo de enteros que tiene la ariedad de la anterior lista de funciones, un arreglo que tiene las terminales que van a intervenir en la función a inducir.

5.1 Funciones a Inducir

Se han inducido dos funciones, así:

1. La función mayor o igual que, con los siguientes parametros:

```
String[][] examples = {
{"geq(0,1)", "false"},
{"geq(0,0)", "true"},
{"geq(1,0)", "true"},
{"geq(1,1)", "true"},
{"geq(1,2)", "false"},
{"geq(2,1)", "true"},
{"geq(2,5)", "false"},
{"geq(5,2)", "true"},
{"geq(3,3)", "true"}
};

String[] functor = {"geq", "s"};

int[] arityFun = {2, 1};

String[] terminal = {"0", "X", "Y", "true", "false"};
```

2. La función máximo entre dos números, con los siguientes parámetros:

```
String[][] examples = {
{"max(0,8)", "8"},
{"max(8,0)", "8"},
{"max(5,8)", "8"},
{"max(8,5)", "8"},
{"max(0,0)", "0"},
```

```
{"max(100,0)", "100"},
{"max(0,100)", "100"},
};

String[] functor = {"max", "s"};

int[] arityFun = {2, 1};

String[] terminal = {"0", "X", "Y"};
```

5.2 Resultados

A continuación se van a presentar las gráficas de los comportamientos del fitness y los operadores a través del proceso de inducción.

geq(,)$_1$:

En la figura 1 se presenta las gráficas del comportamiento del proceso evolutivo de inducir la función **geq(,)**, la función inducida fue la siguiente.

```
geq(Y,Y) = true; geq(X,true) = geq(false,X); geq(s(X),Y) = geq(X,Y);
geq(X,Y) = false;
```

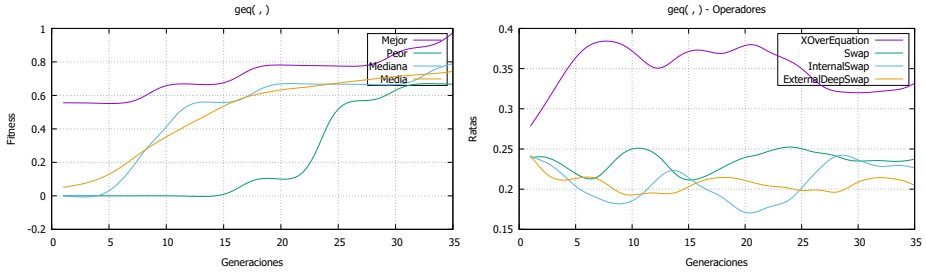

Fig. 1. Del lado izquierdo se presenta la evolución de la función de fitness y del lado derecho el comportamiento de los operadores genéticos a través del proceso evolutivo

geq(,)$_2$:

En la figura 2 se presenta las gráficas del comportamiento del proceso evolutivo de inducir la función **geq(,)**, la función inducida fue la siguiente.

```
geq(Y,s(Y)) = false; geq(Y,s(s(s(Y)))) = false; geq(X,Y) = true; geq(X,Y) = true;
```

max(,)$_1$:

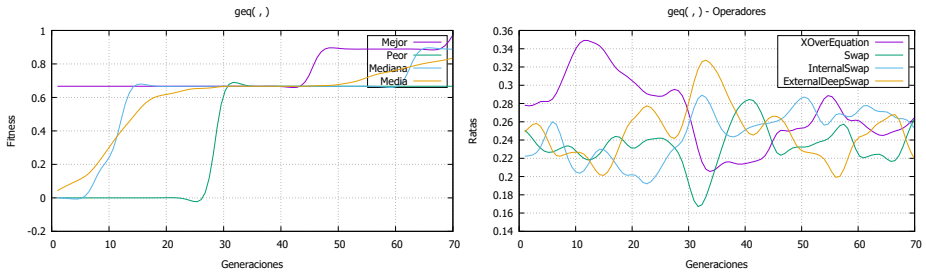

Fig. 2. Del lado izquierdo se presenta la evolución de la función de fitness y del lado derecho el comportamiento de los operadores genéticos a través del proceso evolutivo

En la figura 3 se presenta las gráficas del comportamiento del proceso evolutivo de inducir la función **max(,)**, la función inducida fue la siguiente.

```
max(Y,0) = Y; max(s(X),s(Y)) = s(max(Y,X)); max(max(s(Y),X),s(s(0))) = s(s(s(0)));
max(X,s(Y)) = s(Y);
```

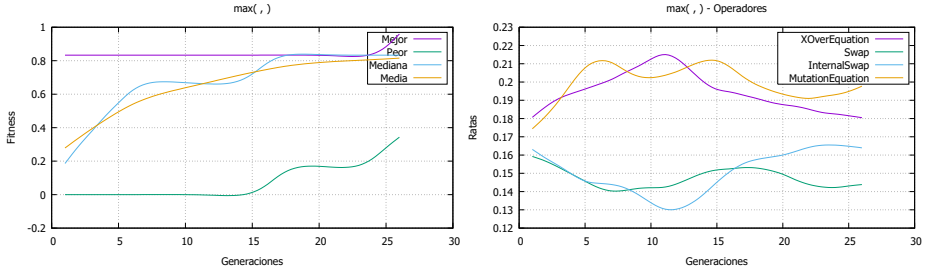

Fig. 3. Del lado izquierdo se presenta la evolución de la función de fitness y del lado derecho el comportamiento de los operadores genéticos a través del proceso evolutivo

max(,)$_2$:

En la figura 4 se presenta las gráficas del comportamiento del proceso evolutivo de inducir la función **max(,)**, la función inducida fue la siguiente.

```
max(s(s(s(Y))),Y) = s(s(s(Y))); max(s(s(s(Y))),Y) = s(s(s(Y)));
max(Y,0) = Y; max(X,Y) = Y;
```

6 Conclusiones

1. Se encuentra que el proceso de generación de población totalmente aleatorio, produce resultados poco convenientes, es necesario estudiar mas a fondo sobre el proceso de la generación de población inicial.

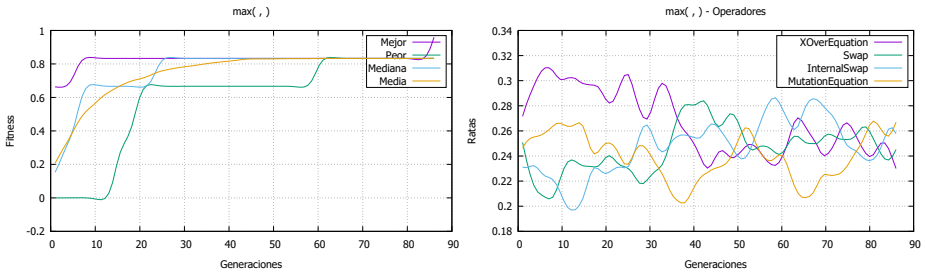

Fig. 4. Del lado izquierdo se presenta la evolución de la función de fitness y del lado derecho el comportamiento de los operadores genéticos a través del proceso evolutivo

2. El proceso evolutivo que esta implicado no fue modificado, en este caso de estudio solo fue necesario producir la arquitectura alrededor de la naturaleza del cromosoma que para este caso son ecuaciones.

Trabajo Futuro

Se recomienda hacer un estudio mas profundo acerca de la estrategia de generación de población inicial, así como el replanteo de los operadores genéticos de mutación, puesto que dada la aleatoriedad de este produce resultados poco deseados.

References

1. Edwin Camilo Cubides Garzón, *Genetic programming*, (2015).
2. Jonatan Gomez, *Self adaptation of operator rates in evolutionary algorithms*, Genetic and Evolutionary Computation–GECCO 2004, Springer, 2004, pp. 1162–1173.
3. John R Koza, *Genetic programming: on the programming of computers by means of natural selection*, vol. 1, MIT press, 1992.