11 "Programación Genética"

Daniel José Abate
danieljoseabate@gmail.com

Resumen - Este trabajo es una Investigación sobre Programación Genética y fué presentado como trabajo final de la materia Introducción a la Inteligencia Artificial. El objetivo está enfocado en realizar una descripción de esta metodología de programación basado en una búsqueda bibliográfica, recaudar y clasificar información disponible sobre sus orígenes, herramientas actuales de desarrollo y sus aplicaciones en la tecnología. Se realiza una descripción teórica general del tema y en particular el análisis de un ejemplo práctico de implementación de una solución al problema de "Regresión Simbólica" utilizando el framework ECJ (A Java-based Evolutionary Computation Research System).

11.1 Introducción

La *Programación Genética* (PG) es una técnica para crear automáticamente programas de computación que solucionan un problema a partir del enunciado del mismo sin la necesidad de programar explícitamente el programa. Su objetivo es dar respuesta a una de las preguntas centrales de la Ciencia de la Computación:

¿ Cómo podrán aprender las computadoras a resolver problemas sin que se las tenga que programar explícitamente ? Arthur Samuel 1959.

En su libro "*Genetic Programming: On the Programming of Computers by Means of Natural Selection"* publicado en 1992 [1], John R. Koza presentó a la PG como un método automático, independiente del dominio, con la capacidad de producir genéticamente programas para resolver una amplia variedad de problemas en una amplia variedad campos. Para esto se cuenta con un conjunto de programas creados inicialmente al azar, llamado *población de programas*. A esta población se le realiza la evolución genética utilizando los principios de la teoría de Darwin y las operaciones inspiradas en la biología tales como la selección natural, la reproducción y la mutación. Iterativamente, la *población de programas* es transformada en una nueva generación de programas luego de aplicar estas operaciones. Los individuos son evaluados en cada iteración para conocer sus características De esta forma en cada nueva generación se consiguen individuos más capaces de solucionar el problema planteado.

11.2 Desarrollo

11.2.1 Descripción Teórica

En el paradigma de Darwin se explica la evolución como un proceso de adaptación al medio ambiente y transferencia genética donde los individuos más aptos tienden a sobrevivir y a reproducirse más. Sus caracterésticas son:

- La gran variedad de organismos existentes evolucionaron de formas más simples.
- La selección natural es probabilística y no determinística.
- Herencia de características a la descendencia con variaciones en la transmisión. Estas son en primera instancia producto de la cruza y en segunda instancia de la mutación.

• La evolución es un cambio a causa de la adaptación y la diversidad, no es simplemente un cambio en los códigos de los genes.

Entonces la evolución puede ser vista como un proceso de aprendizaje, o bien como un proceso de optimización el cual puede ser modelado algoritmicamente y simulado en computadoras donde el modelo más elemental se resume a la Ec. 1.

$$x[t + 1] = v(s(x[t])) \quad (1)$$

La población x en el tiempo t es representada como x[t], sus individuos son elegidos por una selecćon s para luego ser transformados mediante una variación al azar v y dar lugar a una nueva población x[t+1].

La *Computación Evolutiva* (CE) es una rama de la Inteligencia Artificial que utiliza el modelo anterior, ha tenido una intensa actividad de investigación surgiendo así diferentes paradigmas:

- Estrategias Evolutivas (EE)
- Programación Evolutiva (PE)
- Algoritmos Genéticos (AG)
- Programación Genética (PG)

Estos paradigmas utilizan el mismo concepto, ya que tienen en común un proceso que inicia un conjunto de soluciones de prueba aleatorias que conforman la primera *poblacíon de soluciones*, un mecanismo de selección basado en la *aptitud* para determinar a cuales individuos aplicarle la operación de variación al azar y una medición objetiva del desempeño de cada solución para evaluar su aptitud y poder guiar el proceso evolutivo. De todos modos, estos paradigmas dar lugar a modelos de evolución que difieren fundamentalmente en los siguientes cuatro aspectos:

- Perspectiva desde la cual la evolución es modelada.
- El tipo de estructuras empleadas para representar a los individuos de la población.
- La forma en que las estructuras son manipuladas para producir la descendencia.
- La forma de evaluar las estructuras y de seleccionar a los nuevos padres.

La idea del paradigma *Estrategias Evolutivas*, consiste en utilizar eventos estadísticos en una variable de control para producir cambios aleatorios en una variable objetivo, dando lugar a una mutación discreta de las variables objetivo, permitiendo explorar espacios de búsqueda complejos. La versión original de EE denominada como (1+1) – EE utiliza un solo padre y con él se genera, mediante una variación, un solo hijo que se mantiene solo si es mejor que el padre, de lo contrario se descarta (selección extintiva). En la estrategia (μ + 1) – EE se cuentan con μ padres y se genera solo un hijo, el cual puede reemplazar al peor padre de la población. El uso de múltiples hijos se introdujo con las estrategias (μ + λ) – EE y (μ, λ) – EE , donde en la selección (+), los μ mejores individuos obtenidos de la unión entre padres e hijos sobreviven y en la selección (,) solo los μ mejores hijos sobreviven. En EE también se evolucionan las variables de control, lo que permite una auto-adaptación.

El paradigma original de *Programación Evolutiva* (PE), consiste en generar una población de autómatas de estado finito, para luego exponerlos a una determinada cadena de símbolos de entrada con el propósito de que tales autómatas puedan predecir el orden en que aparecen los símbolos. La *aptitud* de cada autómata es en función de que tan bueno es para predecir símbolos. Para dar origen a las nuevas generaciones, los autómatas hijo se generan mediante un mecanismo de copiado de los autómatas padre y luego se aplica el operador de mutación.

Luego de un estudio formal referente al fenómeno de adaptación en sistemas naturales con el objetivo de conocer el proceso evolutivo, se dio lugar a los llamados *planes reproductivos genéticos*, conocidos hoy en día como *Algoritmos Genéticos* (AG). Estos se definen como una búsqueda probabilística que iterativamente transforma un conjunto de objetos matemáticos (población) en un nuevo conjunto de objetos descendientes. Cada objeto tiene asociado un valor de *aptitud* el cual es utilizado para guiar la evolución, junto con el principio de selección natural de Darwin y con las operaciones genéticas ya mencionadas.

Los representación de los individuos es mediante cadenas de longitud fija o variable llamadas *cromosomas*.
El carácter probabilístico de los AG se encuentra claramente fundado en sus pasos ya que, la población inicial es típicamente generada al azar, la selección es probabilística y basada en la aptitud, de esta forma el mejor individuo no siempre es elegido, como del mismo modo el peor individuo no es obligatoriamente excluido, la mutación y los puntos de cruza también son elegidos al azar.
El proceso tiene sus fundamentos matemáticos en el teorema del esquema.
Algunas de sus aplicaciones son optimizacón, aprendizaje, reconocimiento de patrones, bases de datos, planeamiento y predicción.
En la Fig. 1 se muestran los pasos esenciales de un *Algoritmo Genético*.

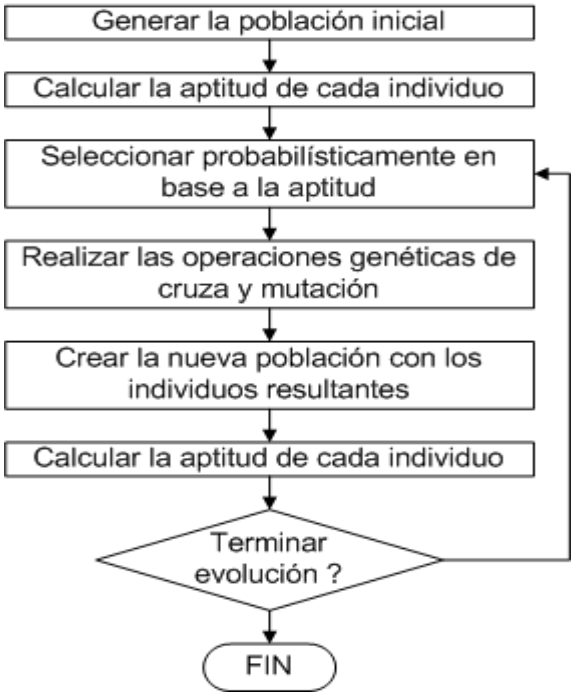

Figura 1: Pasos principales de un *Algoritmo Genético*

El paradigma de *Programación Genética* es una extensión del paradigma de AG con las siguientes diferencias:

- Cada individuo en la población es un programa.
- La representación de los individuos es de alto nivel, como lo son los árboles y grafos.
- Los individuos se construyen con dos clases de símbolos, llamados Terminal y Función.

El conjunto de símbolos *terminal* definen las entradas de datos a los individuos, este está típicamente formado de elementos tales como: identificadores de variables, identificadores de constantes, valores constantes y parámetros de funciones.
El conjunto de símbolos *función está* formado por los operadores, funciones y otros constructores disponibles en el lenguaje de programación empleado para escribir los programas a evolucionar mediante PG. Tienen el propósito de definir las operaciones que el programa realiza sobre sus símbolos *terminales*.

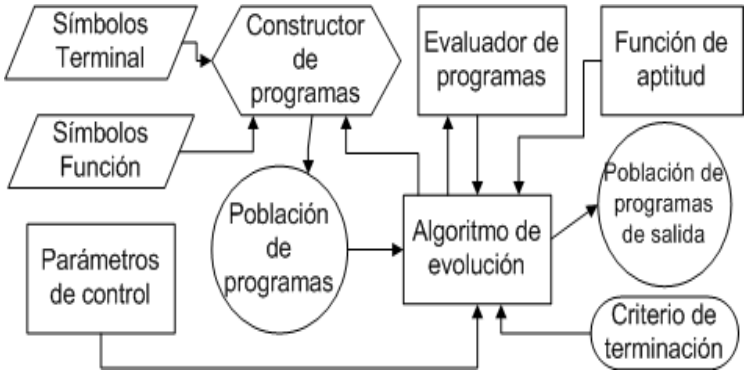

Figura 2: Sistema de Programación Genética

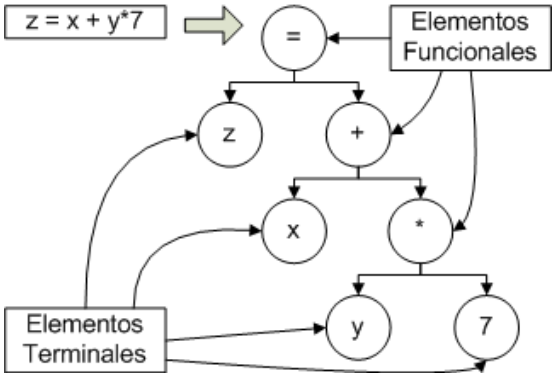

Figura 3: Ejemplo de símbolos terminal y función en PG.

El módulo *constructor de programas* tiene como finalidad construir la población inicial de programas y posteriormente realizar sobre los mismos las variaciones que resulten de la aplicación de los operadores genéticos para dar lugar a una nueva generación de programas.
El *evaluador de programas* es el responsable de ejecutar cada uno de los programas de la población actual para determinar si los resultados obtenidos satisfacen el criterio de terminación de la evolución, encontrando así el programa que mejor resuelve el problema planteado.
La función *de aptitud* asigna un calificación a los programas entregando una retroalimentación continua referente a que tanto un programa es capaz de resolver el problema para el cual fue creado. Típicamente cada programa de computadora es evaluado sobre diferentes *casos de aptitud,* luego su *aptitud* se mide como una suma o un promedio de la misma para diferentes casos representativos. La Ec. 2 es la función *de aptitud* conocida con el nombre de *Distancia de Minkowski* en la cual se realiza la suma de las diferencias absolutas entre las salidas del programa (si) y las salidas esperadas (se). La Ec. 3 es llamada función de e*rror cuadrático.*

$$aptitud = \sum_{i=1}^{n} | si - se | \quad (2)$$

$$aptitud = \sum_{i=1}^{n} (si - se)^2 \quad (3)$$

En el *algoritmo de Programación Genética* existen dos enfoques para implementar el reemplazo de la población actual, el método *generacional* y el de *estado uniforme*. En el enfoque generacional, a partir de la población actual (t) se forma la nueva *población de programas* (t+1), que reemplaza totalmente a la anterior. En el método *de estado uniforme* solo unos pocos individuos son seleccionados y sometidos a los operadores genéticos, de tal forma que los programas hijo resultantes reemplazan a un número igual de programas de la población (tal vez los peores). Como ejemplo se enumeran los pasos básicos de un algoritmo de PG generacional.

1. Crear la población inicial, la generación t=0.
2. Determinar la aptitud de cada programa.
3. Mientras la nueva población de programas no esté completa, hacer lo siguiente:
    1) Seleccionar un(os) programa(s)
    2) Aplicar los operadores genéticos para crear nuevos programas.
    3) Agregar los nuevos programas a la nueva población.
4. Satisfecho el criterio de terminación se detiene el proceso y se ejecuta el paso 5. Caso contrario, se reemplaza la población creando la generación (t=t+1) y vuelve al punto 2.
5. Designar como resultado el mejor de los programas del historial de generaciones.

Los *operadores genéticos* son aplicados a la espera de mejorar la aptitud de los programas.
El *operador de selección* es el encargado de decidir a que individuos aplicar los *operadores genéticos,* a cuales mantener en la población y a cuales reemplazar por los nuevos individuos. Este operador es el responsable de la velocidad con la que avanza la evolución y, de no ser manejado adecuadamente, puede provocar entre otros inconvenientes el de la convergencia prematura. Existen diferentes formas de seleccionar a los individuos. En la Selección *Proporcional* los individuos se eligen en forma estocástica de acuerdo a su contribución de aptitud respecto al total de la población En la Selección *Mediante Torneo* los individuos se eligen mediante comparaciones directas entre ellos. Por último la Selección *de Estado Uniforme* se utiliza en los algoritmos no generacionales donde sólo unos cuantos individuos son reemplazados en cada generación. El *operador de reproducción* realiza una copia idéntica del programa seleccionado y la misma es colocada sin cambios en la nueva población. El *operador de mutación* elige un punto al azar e intercambia este subconjunto de *material genético* (subconjunto de instrucciones) por uno nuevo generado aleatoriamente. El *operador de cruza* permite una alta recombinación del material genético, por lo que es el operador principal de la PG. Generalmente los puntos de cruza no son elegidos con probabilidades uniformes, una estrategia frecuente es elegir a los nodos internos (elementos *funcionales*) el 90% de la veces.

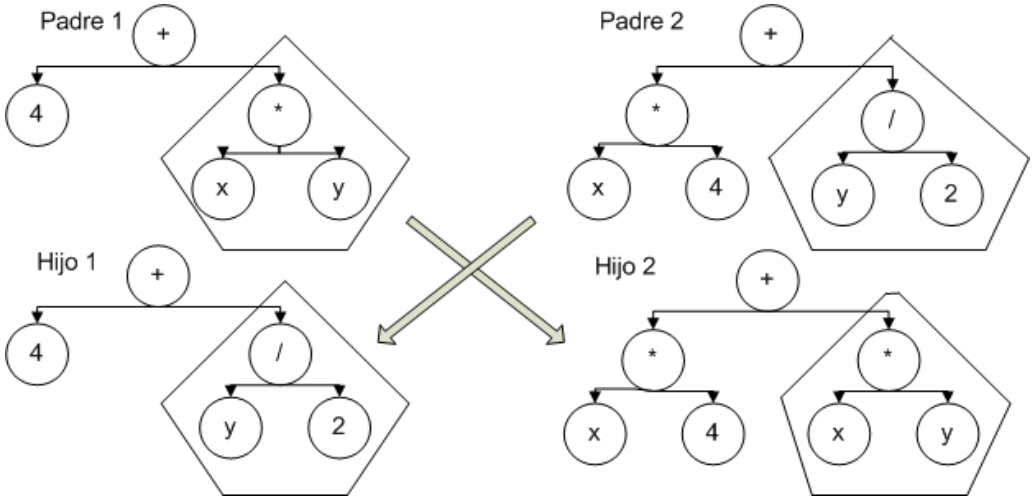

Figura 6: *Operador de cruza*.

11.3 Ejemplo práctico

La *Regresión* es un método matemático que intenta modelizar la relación entre una variable dependiente y variables independientes. En este ejemplo se analizará la solución a la Ec. 4 introducida por John R. Koza en su publicación de 1994. [4]. La Fig. 6 la ilustra.

$$y = x^5 - 2x^3 + x, \{x \in [-1,1]\} \quad (4)$$

Actualmente existen variadas herramientas de desarrollo (*frameworks*) [5] las cuales pueden ser de gran ayuda, brindando un marco conceptual en el aprendizaje. El *framework* ECJ es un sistema de investigación de CE desarrollado en Java [6]. Para la resolución, en primer lugar se define el conjunto de Símbolos *Función* como indica la Ec. 5.

$$SF=\{ + , - , * , /, \sin , \cos , x \} \quad (5)$$

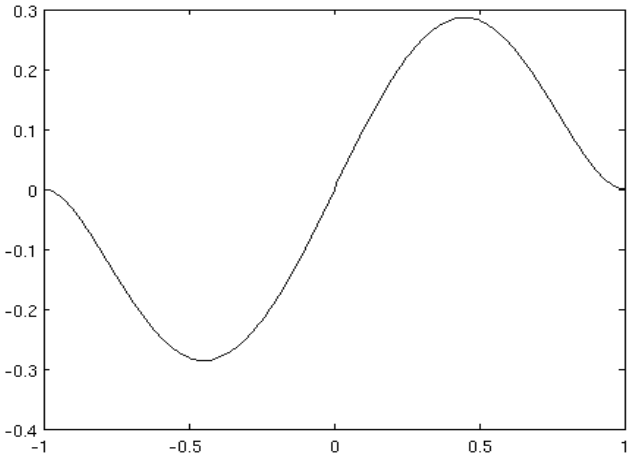

Figura 6: *Función* a descubrir.

Luego se obtienen veinte puntos de la Ec. 4 que se utilizan para evaluar la *aptitud* de los individuos. El *error está* dado por la diferencia entre uno de estos puntos y el resultado del individuo. *Se* considera un éxito (*hit*) cuando el *error* que se está cometiendo al aproximar ese punto es menor a la centésima para este ejemplo. La *aptitud* del individuo se calcula con la *Distanciade Minkowski* expresada en la Ec. 2, esta es la sumatoria del *error* en cada punto siendo el valor 0.0 el mejor individuo, contrariamente el infinito es peor que el peor individuo. También se define la *aptitud ajustada* en la Ec. 6. En este caso el mejor individuo obtendrá 1.0 y el peor se aproximará tanto como sea posible al cero [14].

$$\textit{aptitud ajustada} = 1 / (1+\textit{aptitud}) \qquad (6)$$

El *operador de selección* utilizado es del tipo *Mediante Torneo* con 7 individuos concursantes.
El *operador de cruza* es aplicado con una probabilidad de 0.9, siendo 0.1 la probabilidad de aplicar el *operador de reproducción.* Para la selección del punto se utiliza la estrategia de elegir Símbolos *Funcionales* el 90% de las veces, 10% Símbolos *Terminales* y nunca se eligirá el nodo raíz. La cantidad de generaciones a evolucionar es 50.
En la Fig. 7 se ve la característica de la evolución, donde se grafica el mejor individuo de cada generación. Esta revela que el mejor individuo del proceso aparece en la generación 39, este tiene una *aptitud* de 0.09322218, una *aptitud ajustada* de 0.91472715 y 18 de los 20 puntos evaluados dentro de la zona de éxito. La Ec. 7 lo describe y la Fig. 9 lo representa. El individuo conseguido es bueno, no solo por que su *aptitud ajustada* se encuentra a menos de 9 centésimas de la unidad, sino también porque haciendo un análisis de la Ec. 4 y de la Ec. 7 en un mismo gráfico se ve que en la zona de máximo error, este es menor a 5 centétsimas. Esto se indica la Fig. 8.
ECJ permite obtener datos importantes sobre los tiempos y cantidad de memoria utilizados para la ejecución del programa. La Tabla 3 muestra los tiempos y la Tabla 4 el uso de memoria tomados de un Netbook Acer 751h con un procesador Intel Atom Z520 (1.33Ghz) y Ubuntu 9.10.

Operación	Tiempo	Nodos	Tasa [N/s]
Inicialización	0,46s	5939	12882,86
Evaluación	2,629	623868	237302,41
Cruza	2,83s	617929	217887,52

Tabla 3: Tiempos de ejecución.

Operación	Memoria	Nodos	Tasa [N/Kb]
Inicialización	621 Kb	5939	9,55
Cruza	25843 Kb	617929	23,91

Tabla 4: Uso de memoria.

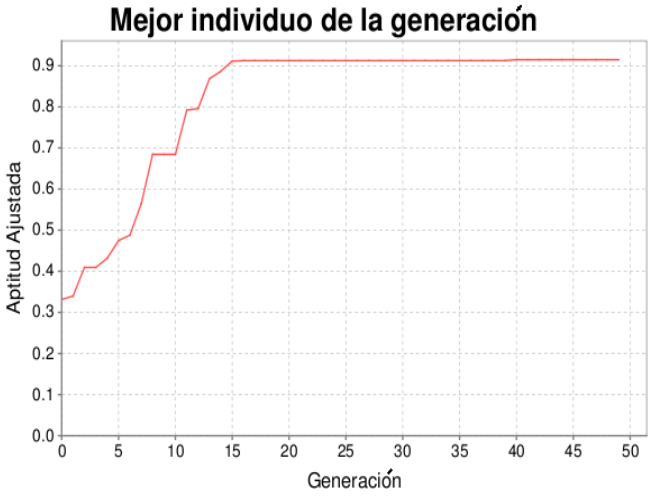

Figura 7: Aptitud del mejor individuo de la generación.

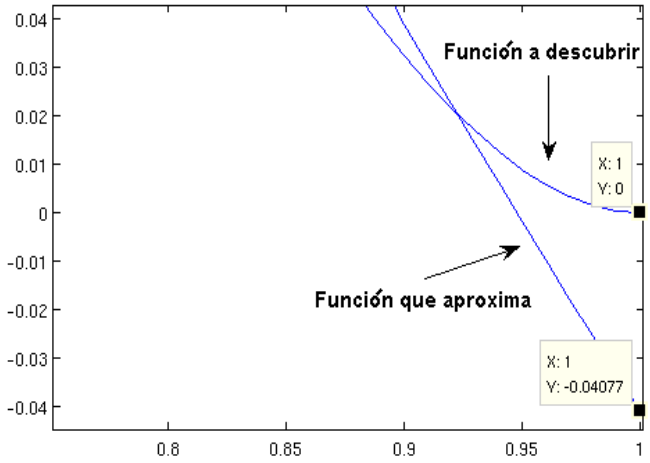

Figura 8: Zona de máximo error.

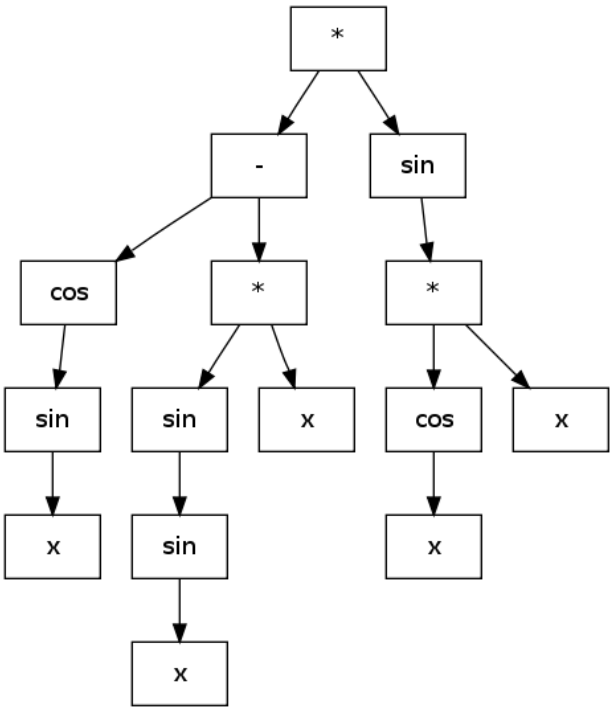

Figura 9: Mejor Individuo.

$$y = (\cos(\sin(x)) - (\sin(\sin(x)) * x)) * \sin(\cos(x) * x), \quad (7)$$

11.4 Conclusiones

La *programación genética* es una metodología con un gran potencial, actualmente se ha creado una máquina con capacidad inventiva [10]. Se trata de un artefacto que puede realizar en paralelo hasta 1.000 procesamientos, a partir de los cuales genera nuevos inventos patentables.
Uno de los más destacados es una antena que ya fue utilizada por la NASA en uno de sus microsatélites experimentales, y que se ha convertido en el primer objeto ideado artificialmente en ser lanzado al espacio [9]. Además de la antena, hay treinta y seis casos donde la PG ha dado resultados competitivos comparados con la performance humana [11].
Con respecto al proyecto ECJ ha cumplido con las expectativas, es robusto, flexible y está ampliamente documentado posicionándose como uno de los mejores entornos de desarrollo de CE [13]. Como desventaja se puede mencionar su consumo de memoria. Como se ve en la Tabla 4 se utilizan en promedio 30 bytes por nodo, comparándolo con Lil-gp [12], donde solo se utilizan 4 bytes por nodo. Por este motivo, en la práctica, un programa que utiliza 8Mb en Lil-gp, utilizará 64Mb en ECJ. Esto es debido en parte a la plataforma Java, y en parte a que ECJ está implementado con árboles en lugar de vectores, lo que logra mayor velocidad de ejecución y flexibilidad para realizar modificaciones. También hay que destacar al proyecto JGAP [7], este está basado en árboles pero implementado con vectores, es menos potente que ECJ pero tiene una curva de aprendizaje más rápida. Por último, mencionar el proyecto WatchMarker [8], este ha dado buenos resultados en comparativas de implementaciones del mismo problema respecto de los dos *frameworks* anteriores y si bien se encuentra en una fase temprana de su desarrollo (versión 0.7), promete en un futuro estar entre los mejores [15][16].

11.5 Referencias

1 - *Genetic Programming: On the Programming of Computers by Means of Natural Selection,* John R. Koza, 1992.
2 - http://www.genetic-programming.com
3 - *Notas Sobre PG y CE*, Perfecto Malaquías Quintero Flores, Instituto Tecnológico de Apizaco.
4 - *Genetic Programing II: Automatic Discoveryof Reusable Programs,* John R. Koza, 1994.
5 - http://en.wikipedia.org/wiki/Genetic_programming#Implementations
6 - http://cs.gmu.edu/~eclab/projects/ecj/
7 - http://jgap.sourceforge.net/
8 - http://watchmaker.uncommons.org/
9 - http://www.nasa.gov/centers/ames/research/exploringtheuniverse/borg.html
10 - http://www.tendencias21.net/Crean-unamaquina-inventora-basada-en-programaciongenetica_a966.html
11 - www.geneticprogramming.com/humancompetitive.html
12 - http://garage.cse.msu.edu/software/lil-gp/
13 - http://userweb.port.ac.uk/~khourym/tutorials.html#1
14 - http://www.genetic-programming.com/jkpdf/tr1314.pdf , página 10, inciso 3.4.
15 - http://blog.uncommons.org/2009/09/17/evolutionary-computation-in-java-ecj-jgap-and-watchmaker-compared/
16 - http://cvalcarcel.wordpress.com/2009/09/13/watchmaker-a-first-and-second-example/