CAMPUS
DE EXCELENCIA
INTERNACIONAL

Graduado en Ingeniería Informática

Universidad Politécnica de Madrid

Facultad de Informática

TRABAJO FIN DE GRADO

Programación Genética Guiada por Gramáticas

Autor: Hugo Domínguez Sanz

Director: Juan Pedro Çaraca Valente

MADRID, JUNIO DE 2013

A la abuela Paz, al abuelo Antonio,

a la Nines y al abuelo Pedro.

AGRADECIMIENTOS

Me gustaría mostrar mi agradecimiento a D. Juan Pedro Çaraca Valente por haberme ayudado y guiado en esta última etapa en la Universidad, guiándome en todo el desarrollo del presente proyecto.

También, dar las gracias a Dª. Aurora Perez Pérez, por el interés mostrado, ya que en ocasiones nos ha dado un punto de vista diferente a algunos conceptos.

También agradezco la ayuda de mis compañeros de la universidad: Miguel Rodriguez Bueno, Borja Marabini Vega, Isidro Asenjo Pessino, Alicia Doblas Alaez y Borja Blázquez Sánchez, con los cuales he compartido muchísimas horas discutiendo sobre la vida, futbol y tecnología en el club Deportivo.

Gracias a toda la gente que he conocido durante estos años en la Universidad, con los que puedo decir que son amigos verdaderos, con lo que espero seguir manteniendo el contacto durante los años próximos y el resto de nuestras vidas.

También me gustaría dar las gracias a mis compañeros del Elsa Pataky, con los que he compartido buenos ratos jugando al futbol en la Facultad, y también a mis compañeros de la selección de la Facultad de Informática, con los que he llevado el nombre de la Facultad al lugar donde se merece en el terreno deportivo, cosechando títulos en Futbol 7, Futbol sala y Futbol 11.

Por último y no menos importante, quiero dar las gracias a aquellas personas que más me quieren: mis padres, mis abuelos, mis tíos y mis primos, sin los cuales llegar hasta aquí, hubiese sido imposible. Muchas gracias de corazón por apoyarme durante todo este tiempo. Este trabajo está especialmente dedicado a vosotros, Muchísimas Gracias.

RESUMEN

Este trabajo fin de grado, presenta una herramienta para experimentar con técnicas de la Programación Genética Guiada por Gramáticas.

La mayor parte de los trabajos realizados hasta el momento en esta área, son demasiado restrictivos, ya que trabajan con gramáticas, y funciones fitness predefinidas dentro de las propias herramientas, por lo que solo son útiles sobre un único problema.

Este trabajo se plantea el objetivo de presentar una herramienta mediante la cual todos los parámetros, gramáticas, individuos y funciones fitness, sean parametrizables. Es decir, una herramienta de carácter general, valida para cualquier tipo de problema que sea representable mediante una gramática libre de contexto.

Para abordad el objetivo principal propuesto, se plantea un mecanismo para construir el árbol de derivación de los individuos de acuerdo a una gramática libre de contexto, y a partir de ahí, aplicar una serie de operadores genéticos guiados por gramáticas para ofrecer un resultado final, de acuerdo a una función fitness, que el usuario puede seleccionar antes de realizar la ejecución.

La herramienta, también propone una medida de similitud entre los individuos pertenecientes a una determinada generación, que permite comparar los individuos desde el punto de vista de la información semántica que contienen.

Con el objetivo de validar el trabajo realizado, se ha probado la herramienta con una gramática libre de contexto ya predefinida, y se exponen numerosos tipos de resultados de acuerdo a distintos parámetros de la aplicación, así como su comparación, para poder estudiar la velocidad e convergencia de los mismos.

ABSTRACT

This final project presents a tool for working with algorithms related to Genetic Grammar Guided Programming.

Most of the work done so far in this area is too restrictive, since they only work with predefined grammars, and fitness functions built within the tools themselves, so they are only useful on a single problem.

The main objective of this tool is that all parameters, grammars, individuals and fitness functions, are can be easily modified thought the interface. In other words, a general tool valid for any type of problem that can be represented by a context-free grammar.

To address the main objective proposed, the tool provides a mechanism to build the derivation tree of individuals according to a context-free grammar, and from there, applying a series of grammar guided genetic operators to deliver a final result, according to a fitness function, which the user can select before execution.

The tool also offers a measure of similarity between individuals belonging to a certain generation, allowing comparison of individuals from the point of view of semantic information they contain.

In order to validate the work done, the tool has been tested with a context-free grammar previously defined, and numerous types test have been run with different parameters of the application. The results are compared according to their speed convergence.

ÍNDICE

1-NTRODUCCIÓN

El presente trabajo tiene el objetivo de crear una herramienta para poder analizar el comportamiento de los operadores de la programación genética guiada por gramáticas, así como ofrecer una herramienta para la realización exhaustiva de pruebas con diferentes dominios. La característica principal del trabajo es conocer bien los diferentes algoritmos de la programación genética guiada por gramáticas, y ofrecer las medidas de evaluación para su posterior análisis.

La principal diferencia de esta herramienta con otras creadas con anterioridad reside en que esta herramienta vale para multitud de casos, es decir, podemos analizar todas las gramáticas libre de contexto que queramos y sus individuos de acuerdo a una serie de parámetros. Por lo que los objetivos fundamentales de este trabajo son:

- Realización de un estudio de los diferentes algoritmos, técnicas y métodos conocidos dentro del marco de la programación genética guiada por gramáticas y analizar las dificultades encontradas.
- Analizar los resultados obtenidos en este trabajo, para facilitar la construcción de nuevos sistemas dentro del marco de la programación genética.

Para abordar los objetivos marcados, el trabajo se organiza en varios capítulos, cutos contenidos son los siguientes:

- *Capitulo 2 - Estado del Arte:* Se realiza un pequeño análisis del estado del arte centrando la atención en la programación genética guiada por gramáticas dentro del marco de la programación genética.
- *Capitulo 3 – Conceptos específicos:* Mediante el cual, quedaran explicados con detalle los diferentes algoritmos de la programación genética guiada por gramáticas, así como los conceptos previos necesarios para su correcta compresión.
- *Capitulo 4 – Planteamiento del problema:* En el que se definen de una manera más concreta los objetivos del trabajo.
- *Capitulo 5 – Resolución del problema:* En el que se detallaran los métodos, técnicas y diferentes algoritmos utilizados para resolver el problema propuesto.

- *Capitulo 6 – Resultados y análisis:* Exposición de ejemplos y resultados de la herramienta, así como un análisis exhaustivo de cada ejemplo mencionado.
- *Capitulo 7 – Conclusiones:* Se realizara una evaluación final del trabajo a través del análisis del cumplimiento de los objetivos.
- *Capitulo 8 – Bibliografía:* Donde se recogen referencias a toda la documentación utilizada para el desarrollo del trabajo.

2-ESTADO DEL ARTE

La computación evolutiva pertenece a una rama de la inteligencia artificial que implica problemas de optimización combinatoria, que a su vez, también esta inspirado en mecanismos de la Evoluciona biológica.

En los años 50, se comenzaron a aplicar los principios de Charles Darwin en la resolución de problemas, pero fue durante los años 60 y 70, donde varias corrientes de investigación definieron lo que actualmente conocemos como computación evolutiva:

- Estrategias Evolutivas.
- Algoritmos Genéticos.
- Programación Genética.

En el campo de la informática, las estrategias evolutivas comprenden métodos computacionales que trabajan con una población de individuos que pertenecen a un determinado dominio. En este caso, cada individuo de la población puede ser un candidato óptimo para la función objetivo. Para representar los individuos de la población, existen 2 tipos de variables:

- Variables objeto: Posibles valores que hacen que la función objetivo alcance el óptimo global.
- Variables estratégicas: Parámetros mediante los cuales se gobierna el proceso evolutivo

Los Algoritmos Genéticos, son un método de búsqueda dirigida basada en probabilidades. Bajo una condición muy débil, podemos demostrar que el algoritmo converge en probabilidad al óptimo.

Los algoritmos genéticos, pueden presentar múltiples variaciones, que dependen de cómo son aplicados los distintos operadores genéticos (cruce y mutación), de cómo es realizada la selección de individuos y de cómo se realiza el remplazo de dichos individuos para formar la nueva población. En general, un algoritmo genético básico, funciona de la siguiente manera:

- **Inicialización**: Aleatoriamente, se genera la población inicial, que esta constituida por una serie de individuos los cuales son posibles soluciones al

problema. En el caso de que la población inicial, no se genere aleatoriamente, es importante garantizar que este conjunto de individuos tenga una diversidad estructural para así poder obtener una representación de la mayor parte de la población, para así poder evitar la convergencia prematura.

- **Evaluación**: A cada individuo de la población se le aplicara una función Fitness para saber cómo es de "bueno" dicho individuo.
- **Condición de parada**: El algoritmo genético tendrá que detenerse cuando se alcance la solución óptima, la cual, generalmente es desconocida, por lo que son necesarios otros criterios de detección. Se suelen usar dos criterios:
    - Definir un número máximo de generaciones.
    - Detenerlo cuando no haya cambios entre dos generaciones.

  Mientras no se cumpla la condición de parada, el algoritmo genético realiza lo siguiente:

    - **Selección:** Después de aplicar la función Fitness a cada individuo, y saber como de "bueno" es cada uno, se procede a elegir los individuos que serán cruzados en la siguiente generación. Es evidente, que los individuos que tengan mejor aptitud tiene mayor probabilidad de ser seleccionados.
    - **Cruce:** La recombinación es el principal operador genético, representa la reproducción sexual, ya que opera sobre dos individuos a la vez para generar dos descendientes donde se combinan las características de ambos individuos padres.
    - **Mutación:** Modifica al azar parte de los individuos, para así poder alcanzar zonas de espacio de búsqueda que no habían sido cubiertas por los individuos de la población inicial.
    - **Reemplazamiento:** Una vez, han sido aplicados los operadores genéticos, se seleccionan los mejores individuos, para así poder generar la población de la generación siguiente.

La programación genética es una metodología basada en los algoritmos evolutivos y además está inspirada en la evolución biológica, para poder desarrollar automáticamente programas que realicen una tarea definida por el usuario. Es una especialización de los

algoritmos genéticos, y a su vez es una técnica de aprendizaje automático, utilizada para optimizar una población de programas de acuerdo a una función fitness.

En este tipo de metodología, cabe destacar que la generación de la población inicial es completamente aleatoria, ya que el ordenador, no ha sido programado para generar la población inicial. Se parte de una población inicial de individuos, al que se le aplican sucesivamente los operadores genéticos (selección, cruce, mutación y reemplazamiento) hasta llegar a la solución final del problema

La principal diferencia de este tipo de metodología con los algoritmos genéticos, reside en la preparación preliminar para aplicar el modelo. En los algoritmos genéticos, establecemos un esquema valido para poder representar los diferentes individuos, normalmente una cadena de caracteres de la misma longitud, y se opera sobre esa estructura para así poder llegar a una solución. Mientras que en la programación genética, escogemos los elementos que conformaran los individuos (las funciones, terminales) para operar sobre esas funciones y terminales para poder encontrar una estructura que represente una buena solución al problema.

La programación genética Guiada por Gramáticas aparece como una rama de los sistemas tradicionales de Programación Genética. La principal diferencia reside en que en este caso, se utilizan gramáticas libres de contexto para establecer la definición formal de las restricciones sintácticas del problema planteado. Este tipo de formalismos es necesario, ya que ayuda a resolver el problema del cierre [Koza 92], para que al aplicar un operador de cruce entre dos individuos validos genere también individuos validos, ya que si generamos individuos que no pertenecen a la gramatical, entorpecería demasiado la velocidad de convergencia.

La programación genética guiada por gramáticas ha demostrado un gran rendimiento en problemas con dominios estructurados y se ha convertido en una de las áreas más prometedoras dentro de la investigación de la programación genética.

Si comparamos la programación genética guiada por gramáticas y la programación genética, podemos observar que utilizan los mismos componentes y operadores

genéticos. Pero también hay que destacar que existen algunas diferencias importantes entre amos sistemas.

- Una de las grandes diferencias, aparece a la hora de generar de forma aleatoria la población inicial, ya que los individuos que forman la población inicial en la programación genética guiada por gramáticas tienen que ser palabras aceptadas por la gramática libre de contexto. Por lo que los métodos utilizados para inicializar la población en la programación genética no son validos para la programación genética guiada por gramáticas ya que los individuos se generan aleatoriamente a partir de funciones y de terminales, que nos lleva a generan individuos que no pertenecen a la gramática libre de contexto, con lo que se impide la convergencia hacia la solución buscada.
- La otra gran diferencia la encontramos al aplicar el operador de cruce, ya que los operadores de cruce de la programación genética, no aseguran la generación de descendientes validos con respecto a la gramática libre de contexto.

3-CONCEPTOS ESPECÍFICOS

El presente capitulo, está organizado en 5 secciones. La primera de estas secciones, nos dará una pequeña visión de la computación evolutiva. En la segunda sección se aborda la temática de los algoritmos genéticos, para continuar con la programación genética, y así continuar en la siguiente sección con la programación genética guiada por gramáticas. Finalizando por una explicación de la distancia mínima de edición utilizada para el desarrollo de la herramienta. Apuntar que todos los contenidos de este capitulo, han sido extraídos de la Tesis Doctoral de Agustín Santamaría Falcón, 2011.

3.1 COMPUTACIÓN EVOLUTIVA

En esta sección describiremos las técnicas que forman parte del campo de la Computación Evolutiva (CE). Todas estas técnicas de la CE esta basadas en el modelo de la evolución natural, por lo que comparten una serie de características comunes a todas ellas. A continuación, se nombran las de mayor importancia:

- Partiendo de un conjunto de individuos, utiliza una estrategia de aprendizaje colaborativo. Con cada individuo se representa o codifica un punto del espacio de búsqueda de soluciones en un problema dado. Cada individuo, incorpora información adicional que permite llegar a la solución final del problema.
- A partir del conjunto inicial de individuos se generan sucesivas generaciones. Estas descendencias, obtenidas a partir de los individuos que conforman la población inicial, son generadas de forma pseudo-aleatoria mediante procesos de cruce y recombinación donde la información es intercambiada entre dos o más individuos pertenecientes a la generación anterior o a la generación inicial.
- Evaluando los individuos, se consigue una medida de adaptación (función fitness) de cada uno de ellos su entorno. De acuerdo a esta medida de adaptación, el proceso de selección favorece mas a los individuos que mejor están adaptados, que serán los seleccionados para la reproducción de una forma más frecuente.

Desde un primer momento, las principales ideas de las CE se orientan hacia cuatro objetivos:

- *Optimización*. La evolución en si es un proceso de optimización en el que el objetivo es la búsqueda de la adaptación de dichos individuos al entorno en que habitan.

- *Sistemas robustos*. Los problemas del mundo real, casi nunca son estáticos y los problemas de optimización temporal son cada vez más comunes. Estas circunstancias requieren un cambio en la estrategia que se aplica para poder resolver el problema planteado. El resultado es un procedimiento robusto que tiene la capacidad de ajustar el rendimiento basándose en la realimentación de la salida del sistema.

- *Inteligencia Artificial*. El área de inteligencia artificial, además de las alternativas clásicas, tienen una nueva posibilidad: simular la evolución para construir algoritmos predictivos.

- *Vida Artificial*. En el campo de la biología, se intenta capturar la esencia de la propia evolución en una simulación por computador y usar entonces esta simulación para conseguir una nueva información sobre la física de los procesos de la evolución natural.

3.2 ALGORITMOS GENÉTICOS

Los algoritmos genéticos (AAGG) son un tipo de algoritmos evolutivos utilizados para resolver problemas de búsqueda y optimización. Están basados en la imitación del proceso natural de la evolución para resolver los problemas de adaptación al medio.

Los algoritmos genéticos, se utilizan para simular, mediante generaciones de poblaciones de individuos, la evolución sufrida a través de distintos operadores. Los individuos que conforman parte de una población se representan mediante un conjunto de parámetros denominados genes. Cada uno de los posibles valores que puede tomar

un determinado gen, se denomina alelo. Los genes a su vez, se agrupan para formar cromosomas.

En cada generación, los genes de los individuos de mayor adaptación al medio, serán heredados por un número mayor de individuos. La combinación de estas características, sirve para generar una descendencia altamente adaptada al medio. Pero también existe la posibilidad, que esta combinación, pueda generar genes de individuos peor adaptados al medio, por lo que estos individuos tenderán a desaparecer.

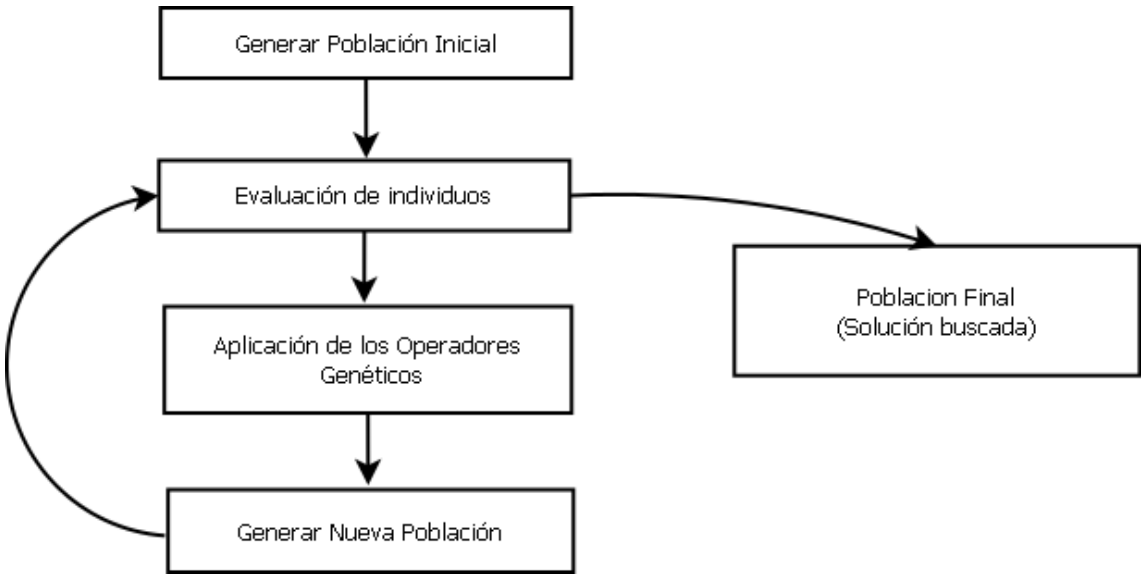

Figura 3-1: Esquema general de funcionamiento de los algoritmos genéticos.

Los algoritmos genéticos, comienzan con una muestra elegida de manera aleatoria en el espacio de soluciones, para así, poder ir transformándola paso a paso, hasta llegar al estado final. Para ello, los algoritmos genéticos, se valen de la acción de los llamados operadores genéticos:

- *Selección*: Elije entre los individuos de una misma población, aquellos que están mejor adaptados al medio.

- *Operador de cruce*: Este operador permite renovar la población. El cruce actúa según cierta probabilidad, obteniendo nuevos individuos, a los que se les llama descendencia, con características obtenidas de sus antecesores.

- *Operador de mutación*: Mediante este operador, se puede modificar de forma aleatoria alguno de los valores de los individuos, lo que nos permite explorar nuevas regiones del espacio de soluciones, que de otro modo serian imposibles de alcanzar.

- *Función de evaluación*: Devuelve un valor de ajuste o de adaptación, que es proporcional a la habilidad de dicho individuo. Esta función, tiene que ser diseñada en función del problema que se quiere resolver.

- *Operador reparador*: Es utilizado para reconstruir aquellos individuos que no verifican las restricciones.

Si el algoritmo genético, ha sido planteado correctamente, la población evolucionara a través de sucesivas generaciones de tal forma que en cada generación, la adaptación del mejor individuo de la población y la media de todos los individuos de cada generación se incremente hacia el óptimo global. Este proceso de incrementación uniforme del valor de adaptación es lo que se denomina como convergencia.

También hay que tener en cuenta que tanto la convergencia de los algoritmos genéticos, como su velocidad de convergencia y el error cometido, dependen en gran medida de multitud de parámetros, como por ejemplo, el tamaño de la población, la forma de aplicar los distintos operadores o su elección.

3.3 PROGRAMACIÓN GENÉTICA

La programación genética (PG) es una extensión de los algoritmos genéticos en la que la población genética está formada por programas de computación. Ambos enfoques difieren en la forma de preparación preliminar para aplicar el modelo. Mientras que en los algoritmos genéticos, se establece un esquema adecuado para la representación de los individuos del problema, y se opera sobre esa estructura predefinida para poder llegar a la solución, en la programación genética, se escogen los elementos que conforman los individuos y se opera con esos elementos para encontrar una estructura que represente una solución adecuada al problema.

El objetivo de la programación genética, es construir programas de computador sin que ellos sean diseñados y programados explícitamente por un humano. Debe aclararse que la creación de estos programas es completamente aleatoria, es decir, el ordenador no ha sido programado para encontrar una solución si no, que esa solución, es alcanzada partiendo de un estado inicial, y aplicando mecanismos de selección a las estructuras intermedias que se van encontrando.

Una vez se ha generado la población inicial de manera aleatoria, la programación genética tiene un funcionamiento similar al de los algoritmos genéticos, aplicando distintos operadores genéticos a los individuos (que son representados mediante arboles), hasta llegar a la solución deseada para el problema. Entre los operadores más utilizados en la programación genética están los siguientes:

- ***Operadores de selección***: son utilizados para elegir en cada generación, quienes serán los individuos sobre los que actuara el algoritmo.

    - *Método Torneo* [Brindle 91]: este método consiste en dividir en una población de N individuos, en conjuntos de K individuos, para así elegir a los individuos mejor adaptados. Cada uno de los ganadores de los diferentes torneos conformara una población intermedia a cada generación, mediante la cual se realizara la reproducción

    - *Método de la ruleta* [Baker 87]: Este método emplea un círculo divido en sectores que son proporcionales a la función objetivo. De esta forma, los individuos mejor adaptados, tienen más posibilidades de ser escogidos, mientras que los individuos peor adaptados, tienen menos probabilidades de ser escogidos para la reproducción.

    - *Método de truncamiento* [Crow y Kimura 70]: Mediante este método, los distintos individuos que conforman la población, son ordenados de acuerdo a su grado de adaptación y solo se seleccionan para la reproducción aquellos que están mejor adaptados. El parámetro que

recibe este algoritmo es el umbral de truncamiento, el cual indica el porcentaje de población que es seleccionada como padre.

- *Método generacional* [De Jong 75]: En este método, se escoge a toda la población para la reproducción. De esta forma, todos los individuos que conforman la población, tienen la oportunidad de transmitir su carga genética a los futuros descendientes, si bien también hay que destacar, que los individuos generados de los antecesores peores adaptados, suelen dar descendientes peor adaptados al medio.

- **Operadores de cruce:** Determinan como se obtienen nuevos individuos a partir de sus individuos progenitores.

  - *Koza* [Koza 92]: El cual, escoge aleatoriamente un nodo de cruce para el primer individuo, otro nodo de cruce para el segundo individuo e intercambia dichos nodos. La parte negativa de este algoritmo, es que provoca un aumento de la complejidad y el tamaño de los individuos, dado que el espacio de búsqueda es potencialmente ilimitado, y los individuos pueden crecer en tamaño durante el proceso de evolución, lo que genera un coste computacional muy elevado, afectando a la velocidad de convergencia del sistema.

  - *SCPC* (Strong Context Preservative Crossover) [D'haeseleer 94]: Este algoritmo escoge aleatoriamente un nodo de cruce del primer individuo, obteniendo su coordenada. Si esta no existe en el segundo individuo, se vuelve a escoger otra coordenada aleatoriamente, pero si la coordenada existe también en el segundo padre, entonces se efectúa el cruce entre los subárboles que cuelgan a partir del nodo de cruce elegido en cada individuo.

  - *Justo* (Fair) [Crawford-Marks y Spector 02]: En primer lugar se escoge aleatoriamente un nodo de cruce del primer individuo, y se calcula la

longitud del subárbol (l), que parte de este nodo de cruce, hasta las hojas. Posteriormente se selecciona un nodo de cruce en el segundo individuo, y se mide la longitud para este árbol (l_2). Si l_2 está dentro del rango [l - 1/4, l+1/4], entonces el nodo de cruce para el segundo padre es aceptado y se procede a intercambiar los subárbols. Si por el contrario l_2, no está dentro de dicho rango, se procede a seleccionar otro nodo de cruce del segundo individuo y se vuelven a realizar las comprobaciones. Si tras n intentos, no se encuentra el subárbol dentro del segundo individuo que satisfaga esa condición, entonces se escoge como nodo de cruce del segundo árbol, aquel subárbol que esté más cerca del rango [l -1/4, l+1/4].

- **Operador de mutación:** Este operador actúa sobre un determinado individuo con una probabilidad p_m. Cuando un individuo, es afectado por este operador, se selecciona aleatoriamente el lugar de mutación. Para ello, cualquier nodo puede ser elegido para realizar la mutación con la misma probabilidad. El subárbol cuya raíz es el nodo elegido para la mutación es reemplazado por un subárbol cuyo símbolo del nodo raíz coincide con el símbolo del nodo escogido para realizar la mutación.

Una vez obtenidos los descendientes de una determinada población, es necesario formar una nueva población. Inicialmente, el proceso de reemplazamiento se efectuaba de uno en uno, por lo que se generaba un único individuo descendiente que sustituía a otro de la población inicial [Hollan 75]. Pero más adelante, se empezó a aplicar otra forma de reemplazar los individuos. Los hijos resultantes reemplazaran a los miembros de la población anterior que peor adaptados se encuentren.

3.4 PROGRAMACIÓN GENÉTICA GUIADA POR GRAMÁTICAS

La programación genética guiada por gramáticas (PGGG), surge como una extensión a los sistemas tradicionales de la programación genética. La principal diferencia, es que se utilizan gramáticas libres de contexto (CFG), para establecer las restricciones sintácticas del problema. El hecho de utilizar estos formalismos sintácticos ayuda a resolver el problema del cierre (*clousure*) [Koza 92], que hace referencia a que el cruce de dos individuos validos, genere también descendientes validos, ya que la generación de individuos inválidos entorpece en gran medida la velocidad de convergencia del sistema. Este tipo de programación genética, ha demostrado ser una aproximación de gran rendimiento para problemas con dominios estructurados y está considerada como una de las áreas más prometedoras dentro de la investigación de la programación genética.

En esencia, la PGGG utiliza los mismos componentes que la programación genética tradicional, aunque existen diferencias de gran importancia entre ambos sistemas. La primera de ellas, es observable a la hora de generar la población inicial, ya que en la PGGG, los individuos que conforman esta población, tienen que ser palabras aceptadas por el lenguaje definido por la gramática libre de contexto en cuestión. Por lo que los métodos utilizados para generar la población inicial en la programación genética, no son validos para la programación genética guiada por gramáticas. Otra diferencia fundamental, aparece a la hora de aplicar los operadores de cruce, puesto que los operadores de cruce tradicionales de la PG no aseguran la generación de descendencia valida con respecto a la gramática.

A continuación, se explica porque estos operadores de cruce, descritos en el apartado anterior, no son validos para la PGGG. Para ello se utilizara la gramática definida en la tabla [3-1].

$G_{ejemplo} = (\sum_N, \sum_T, E, P)$
$\sum_n = \{E, A, D, prB, prS, prP, prH, psS, psB, psP, psH, C, S, B, P, H\}$
$\sum_T = \{c\text{-}agu, c\text{-}mes, c\text{-}irr, s\text{-}agu, s\text{-}sua, b\text{-}agu, b\text{-}sua, p\text{-}gra, p\text{-}peq, h\text{-}gra, h\text{-}peq\}$
P = {
E → A C D
A → prS S psS | prS S | S psS | S
D → prB B psB | prB B | B psB | B
prS → prB B| prP P | prH H | B | P | H
prB → prS S| prP P | prH H | S | P | H
prP → prB B| prS S | prH H | B | S | H
prH → prB B| prP P | prS S | B | P | S
psS →B psB | P psP | H psH | B | P | H
psB → S psS | P psP | H psH | S | P | H
psP → B psB | S psS | H psH | B | S | H
psH → B psB | S psS | P psP | B | S | P
C →c-agu | c-mes | c-irr
S → s-agu | s-sua
B → b-agu | b-sua
P → p-gra | p-peq
H → h-gra | h-peq}

Tabla 3-1: Definición de la Gramática de ejemplo

- El operador de *Koza* elige aleatoriamente los nodos de cruce de los padres, por lo que, a menos que de la casualidad de que ambos nodos de cruce estén etiquetados con el mismo símbolo de la gramática, los individuos que se obtengan, no serán validos.

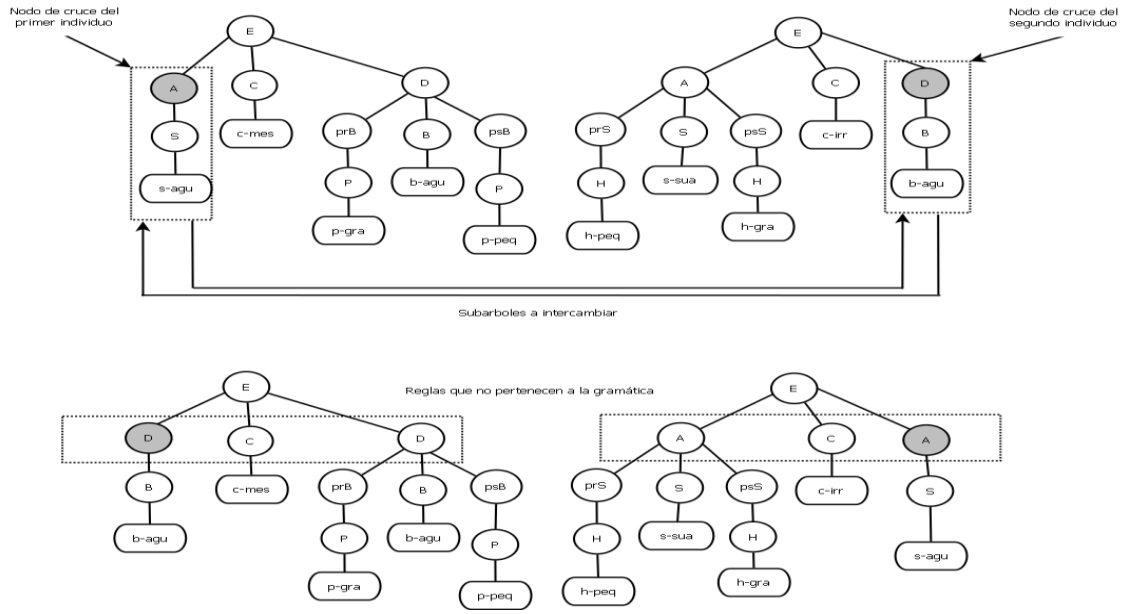

Figura 3-2: Dos individuos generados por la gramática de ejemplo y los arboles resultantes sintácticos resultantes al aplicar el operador de Koza, que son sintácticamente incorrectos.

- El operador *SCPC* exige que ambos nodos de cruce tengan las mismas coordenadas, pero no exige que ambos nodos estén etiquetados con el mismo símbolo gramatical, con lo que es fácil que pueda producir individuos no validos, es decir, que no pertenezcan al lenguaje generado por la gramática. En la imagen [3-3] se muestra un ejemplo de aplicación de este operador utilizando la gramática definida anteriormente.(Tabla[3-1])

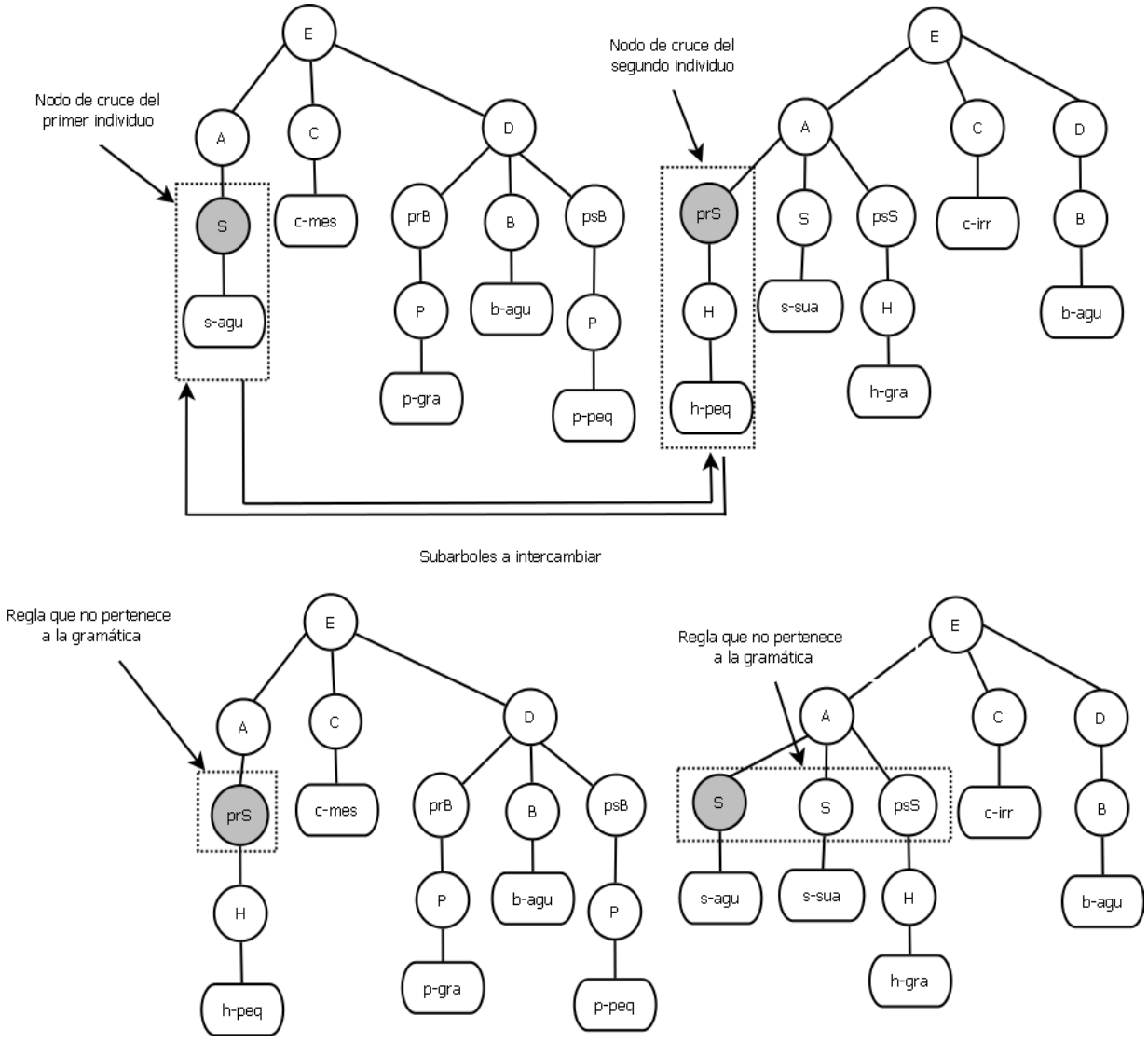

Figura 3-3: Dos individuos generados por la gramática de ejemplo y los arboles resultantes sintácticos resultantes al aplicar el operador de SCPC, que son sintácticamente incorrectos.

- El operador de *Justo* (Fair), presenta el mismo inconveniente que los anteriores al aplicarlo a los sistemas de PGGG, ya que no se asegura que la descendencia obtenida sea sintácticamente valida. La figura ---- constituye un ejemplo en el que los descendientes obtenidos son inválidos a pesar de sustituir subárboles de la misma longitud. La figura 3-2, también podría servir de ejemplo para ver que este operador de cruce, genera descendientes que están mal formados sintácticamente.

Los operadores de cruce, que si generan individuos validos, es decir, individuos que si pertenecen a la gramática serian los siguientes:

- *WX* (*Whigham*) [Whigam 95]: asegura la generación de individuos validos que pertenecen al lenguaje generador por la gramática dado que se cruzan nodos etiquetados con el mismo símbolo no terminal. Este operador consta de los siguientes pasos:

    1. Se almacenan las coordenadas de todos los nodos no terminales del primer padre.

    2. Se elige aleatoriamente uno de ellos, y se anota su símbolo no terminal.

    3. Se almacenan las coordenadas de todos los nodos del segundo padre, cuyo símbolo no terminal, coincida con el obtenido en el paso anterior.

    4. Se elije aleatoriamente una de estas coordenadas

    5. Se intercambian los subárboles cuya raíz coincida con las dos coordenadas elegidas.

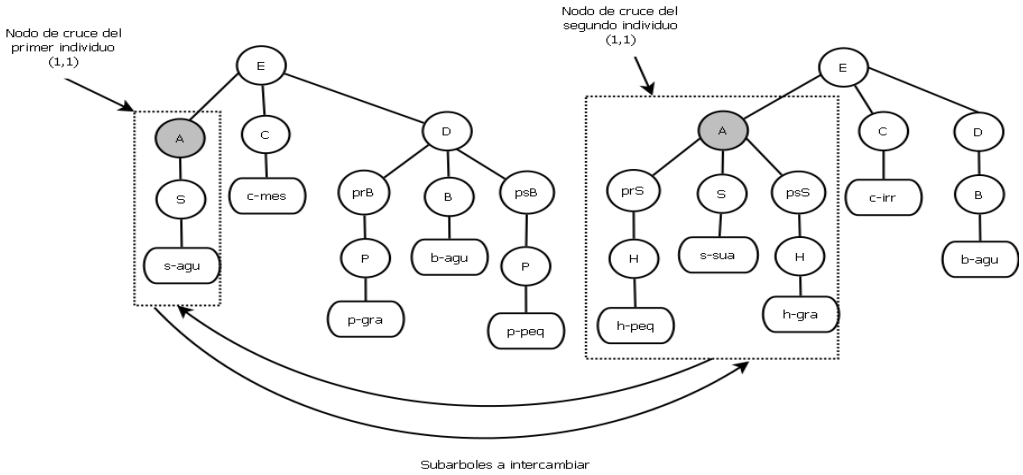

Figura 3-4: Ejemplo de aplicar el operador WX

- *GBC* (*Grammar based Crossover*) [Couchet et al. 07]: El operador GBC evita el crecimiento desmesurado del tamaño de los arboles de derivación que representan a los individuos. También proporciona el equilibrio adecuado entre la capacidad de exploración de espacio de búsqueda y la capacidad de explotación, preservando el contexto, en el cual, los subárboles aparecen en los arboles padre y, finalmente es capaz de aprovechar la propiedad de las gramáticas ambiguas consistente en la existencia de más de un árbol de derivación distinto para una misma palabra. La unión de todas estas características proporciona al operador GBC una elevada velocidad de convergencia y una reducida probabilidad de caída en óptimos locales.

  La aplicación de este operador, consta de los siguientes pasos:

  1- Creación de un conjunto NT compuesto por los nodos no terminales del primer padre (exceptuando el nodo raíz). Cada nodo, queda denotado, utilizando la notación para coordenadas del operador de cruce SCPC, mediante la tupla N = (Z, coord.), donde Z es el símbolo no terminal que representa al nodo de cruce y coord. es la coordenada de dicho nodo en el árbol.

2- Si NT ≠ Φ, entonces se escoge aleatoriamente un elemento de dicho conjunto de manera aleatoria, el cual se denomina nodo de cruce o lugar de cruce (NC1). Si NT = Φ entonces el cruce no es posible en ningún otro nodo que el nodo raíz, por lo que se elije como nodo de cruce para ambos arboles, dando a lugar a descendientes idénticos a los padres.

3- Se busca el nodo padre de NC1, el cual será siempre un símbolo no terminal, ya que estamos trabajando con una gramática libre de contexto. Este símbolo, aparece como antecedente de una o varias reglas de la gramática, por lo que se almacenan todos los consecuentes de estas producciones en una lista R.

4- Se denomina derivación principal a la derivación que se aplica para el nodo padre del nodo NC1. Se definen a sí mismo la longitud de la derivación principal (l), como el número de símbolos terminales y no terminales incluidos en el consecuente de la derivación principal, y la posición (p) que ocupa el nodo de cruce dentro de esta derivación principal. Se calcula y se almacena esta tupla T = (l, p, C).

5- Se eliminan de R, todos aquellos consecuentes que no tengan la misma longitud que la derivación principal.

6- Para cada elemento de R, se comparan todos los símbolos con los del consecuente de la derivación principal excepto aquel que ocupa la misma posición (p) que NC1. Posteriormente, se eliminan de R, todos los consecuentes en los que se ha detectado alguna diferencia.

7- Se calcula el conjunto X, formado por todos aquellos símbolos no terminales pertenecientes a los consecuentes almacenados en R, que están en la misma posición (p) que el nodo de cruce NC1.

8- Si X≠ Φ entonces se elige aleatoriamente algún símbolo (SE, símbolo elegido) perteneciente al conjunto X. En una lista llamada NC, se almacenan

todos los nodos candidatos del segundo padre cuyo valor coincida con el valor SE. Si X = Φ entonces, al no existir el nodo de cruce del segundo padre que satisfaga los requisitos de cruce para los símbolos del conjunto X, se saca NC1, del conjunto NT y se vuelve al paso 2.

9- Si NC ≠ Φ entonces se escoge aleatoriamente un nodo de cruce NC2 del conjunto de nodos candidatos NC del segundo padre, pero si NC = Φ entonces se elimina SE del conjunto X y se vuelve al paso 8.

10- Se calcula la profundidad a la que se encuentra el nodo de cruce del primer padre (NC1) y se le suma la longitud del subárbol cuya raíz, es el nodo de cruce del segundo padre (NC2). Esta profundidad obtenida se denomina como P1. Se calcula la profundad a la que se encuentra el nodo de cruce del segundo padre (NC2) y se le suma la longitud del subárbol cuya raíz es el nodo de cruce del primer padre (NC1). Esta profundidad obtenida se denomina P2.

- Si P1 > D o si P2 > D, entonces elimina NC2 del conjunto NC y se vuelve al paso 9. (D es el valor de la profundidad máxima definida para un individuo).

- En otro caso, se comparan los símbolos no terminales de NC1 y NC2,
    - Si NC1 = NC2 entonces los dos nuevos descendientes se obtienen intercambiando los dos subárbols cuyas raíces coincidan con los dos nodos de cruce previamente establecidos (NC1 y NC2).

    - Si NC1≠ NC2 entonces se calcula la derivación generada por el nodo padre de NC2 y se obtiene la posición (p2) del símbolo NC2 en dicha derivación. Tras lo cual se sustituye el

símbolo de la posición p2, por el símbolo NC1. Si la derivación obtenida coincide con algún consecuente de las reglas de la producción de la gramática, entonces los dos nuevos descendientes se obtienen intercambiando los dos subárbols cuyas raíces coinciden con los nodos de cruce NC1 y NC2. En caso contrario, se elimina NC2 de NC y se vuelve al paso 9.

Para poder ver mejor todo el funcionamiento de este operador, se incluye el siguiente ejemplo, basándonos en la gramática definida anteriormente. [Tabla3-1]

A continuación se detalla paso a paso la aplicación del operador GBC, con los individuos de la figura 3-5.

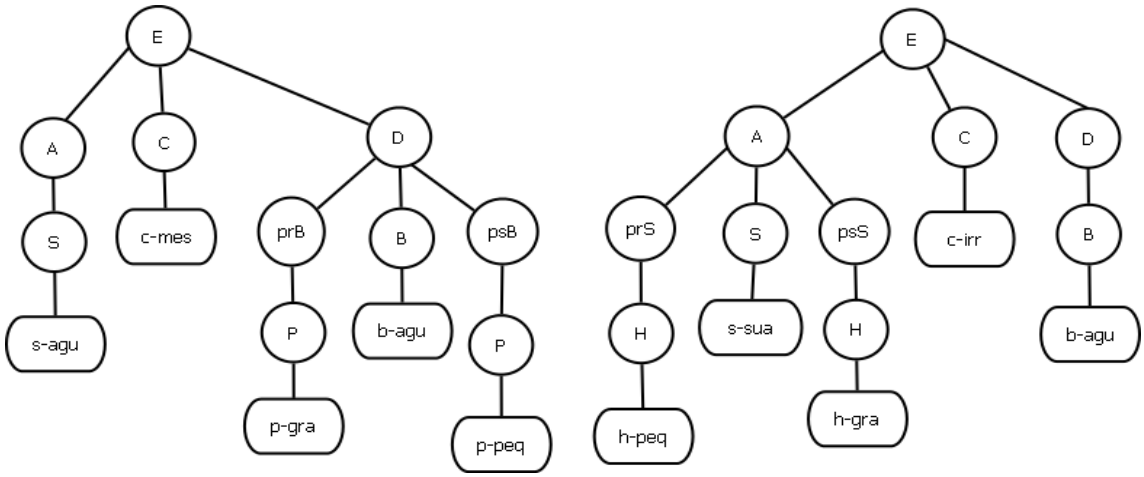

Figura 3-5: Arboles de derivación de dos individuos generados por la gramática de ejemplo

1- Se almacenan para el primer padre, todos los nodos que contienen símbolos no terminales (salvo el nodo raíz) en el conjunto NT.
NT={(A(1)),(C(2)),(D(3)),(S(1,1),(prB(3,1)),(B(3,2)),(psB(3,3)),(P(3,1,1)), (B(3,1,2)) ,(P(3,3,1))}

2- Se elige de manera aleatoria un nodo cualquiera del entre los que componen el conjunto NT.
El nodo elegido es: Nodo de cruce = (S (1,1))

3- Se busca el nodo padre del nodo elegido anteriormente y se almacenan en R todos los consecuentes de las reglas pertenecientes a la gramatical donde "A" aparece como antecedente.
R = [prS S psS | prS S | S psS | S]

4- Se obtiene la derivación principal A → S. La longitud de la derivación principal es, *l = 1*, y la posición que ocupa dentro del nodo de cruce dentro del consecuente de la derivación principal es *p = 1*. Por consiguiente, la tupla obtenida es la siguiente:
T = (1,1, S)

5- Se eliminan de R todos aquellos consecuentes cuya longitud es diferente de la longitud de la derivación principal.
R = [S]

6- Cada elemento de R, se compara con los símbolos del consecuente de la derivación principal excepto aquel que se encuentra en la posición del nodo de cruce. Como consecuencia de esto la lista R quedaría de la siguiente manera:
R = [S]

7- Se calcula el conjunto X, formado por todos aquellos símbolos en los que los consecuentes de R, que están la misma posición, p, que el nodo de cruce. Por lo tanto, se elige para cada elemento de R, el símbolo ubicado en la primera posición.
X = [S]

8- Se elige aleatoriamente un nodo del segundo individuo cuyo símbolo no terminal sea el símbolo elegido S. Para este ejemplo, se escoge aleatoriamente como símbolo elegido (SE) el símbolo S. Se genera el conjunto NC de nodos candidatos del segundo individuo cuyo símbolo no terminal sea S.
NC = {(S (1,2)})

9- La siguiente figura muestra en color gris, el único nodo de cruce posible seleccionado por el operador, ya que el conjunto NC tiene un único elemento.

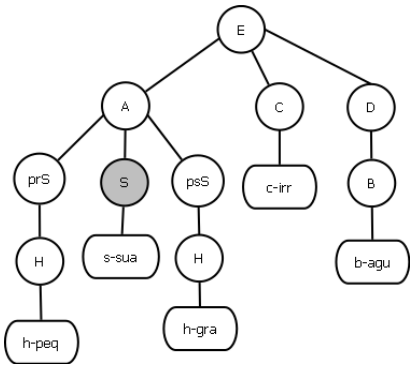

Figura 3-6: Arboles de derivación de dos individuos generados por la gramática de ejemplo

10- Se calcula la profundidad a la que se encuentra el nodo de cruce del primer padre, pf1 = 2, y se le suma la longitud del subárbol cuya raíz es el nodo de cruce del segundo padre, l2 = 3, por lo que P1 = pf1 + l2 = 2+3 = 5. Se calcula también la profundidad a la que se encuentra el nodo de cruce del segundo padre pf2 = 2, y se le suma la longitud del nodo de cruce del primer padre, l1 = 1, por lo tanto P2 = 2+1 = 5. Como la profundidad máxima (D) establecida para los individuos es de 7, ambos descendientes cumplen con la restricción de profundidad, al ser P1≤ 7 y P2 ≤ 7. Como el símbolo no terminal de NC1 y NC2 es el mismo, los dos nuevos descendientes se obtienen intercambiando a los subárboles cuyas raíces coinciden con los nodos de cruce. En la siguiente figura, se muestra el resultado final.

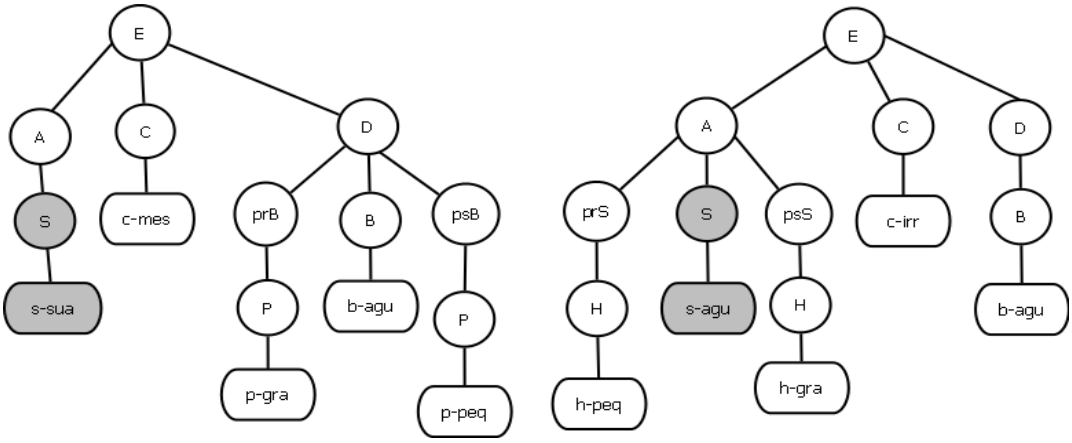

Figura 3-7: Descendencia obtenida al aplicar el operador GBC

Para finalizar, el operador de mutación, el cual debe utilizar producciones de la gramática para así poder asegurar que la mutación del individuo da lugar a otro igualmente valido. Se define el operador de mutación en el ámbito de la PGGG: el operador GBM (*Grammar based mutation*). Con este operador, una vez definido el nodo donde se produce la mutación, el subárbol no es reemplazado por otro cuya raíz coincida con el mismo símbolo no terminal, si no que se extiende su ámbito a otros símbolos no terminales que también generan producciones con respecto a la gramática.

Este operador consta de los siguientes pasos:

1- Se crea un conjunto NT con los diferentes candidatos a nodos de mutación.

2- Si NT ≠ Φ entonces se elige aleatoriamente un elemento de NT, el cual se denomina nodo de mutación o lugar de mutación (NM). Si NT = Φ, entonces la mutación del individuo no es posible, por lo que el individuo mutado es una copia del individuo original.

3- Se busca el nodo padre del nodo de mutación. Este es siempre un símbolo no terminal, puesto que se trabaja con una gramática libre de contexto. Dicho símbolo aparece como antecedente de una o varias reglas de producción de la gramática, por lo que se almacenan en R, todos los consecuentes de estas producciones.

4- Se denomina derivación principal($A \rightarrow C$), a la derivación principal aplicada en el nodo de padre. Se definen a si mismo la longitud de la derivación principal *(l)*, como el numero de símbolos terminales y no terminales incluidos en el consecuente de la derivación principal y la posición (*p*) que ocupa el nodo de mutación en la derivación principal. Se calculan y almacenan estos valores en una tupla T= (l, p, C).

5- Se eliminan de la lista R, todos aquellos consecuentes cuya longitud sea diferente a la longitud de la derivación principal *l*.

6- Para cada elemento de R, se comparan todos los símbolos con los del consecuente de la derivación principal excepto aquel que ocupa la posición *p*. Se eliminan de R todos aquellos consecuentes en los que se ha detectado alguna diferencia.

7- Se calcula el conjunto X, formado por todos aquellos símbolos no terminales pertenecientes a los consecuentes almacenados en R, que ocupan la misma posición p.

8- Si X ≠ Φ entonces se elige aleatoriamente un símbolo (SE) del conjunto X. Si X = Φ entonces no es posible encontrar una mutación para dicho nodo, por lo que se elimina el nodo NM del conjunto NT y se vuelve al paso 2.

9- Se calcula a la profundidad a la que se encuentra el nodo de mutación, y se efectúa la resta entre la profundidad máxima permitida (D), y dicha profundidad. Dicho valor se denomina longitud de la mutación (LM).

10- Se asigna el valor 0 a la profundidad actual (PA).

11- Se almacena en el conjunto PC aquellas producciones de SE, que cumplen la siguiente condición: $PA + L\ (SE ::= \alpha) \leq LM$, donde L (SE ::= α) es la longitud del consecuente de la producción SE ::= α.

12- Si PC ≠Φ, entonces se escoge aleatoriamente una producción del conjunto PC. Si PC = 0, entones no es posible encontrar una mutación a partir del símbolo SE para el nodo de mutación NM que cumpla la condición de profundidad para el individuo mutado, por lo que se elimina SE del conjunto X y se vuelve al paso 8.

13- Por cada símbolo no terminal Beta € alfa se anota B como el actual símbolo SE y se repiten los pasos 11, 12 y 13, incrementando el valor de la profundidad actual en una unidad.

14- Se efectúa el reemplazo del subárbol cuya raíz es el nodo de mutación (NM) por el subárbol calculado en los pasos 11, 12 y 13.

Existen varios tipos de mutación:

- *Terminal* (*mono*): La mutación afecta únicamente a un único nodo que contiene un símbolo terminal.
- *Terminal* (*multi*): La mutación se aplica sobre varios nodos que contienen símbolos terminales.
- *No Terminal*: La mutación está restringida exclusivamente a nodos con símbolos no terminales.

Para poder ver de una manera más clara el algoritmo GBM, se muestra su funcionamiento, con el siguiente ejemplo sobre el individuo representado en la figura --- generado por la gramática de ejemplo.

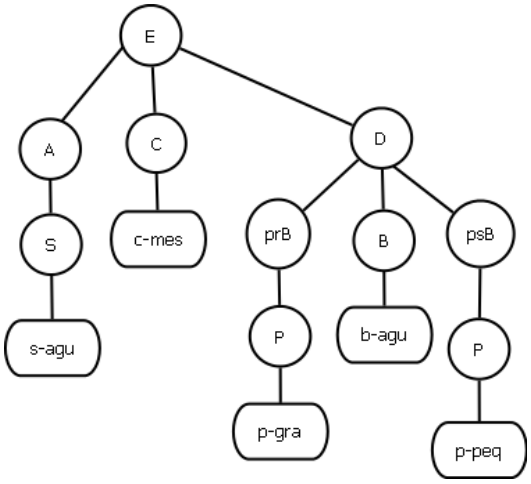

Figura 3-8: Individuo generado por la gramática para realizar la mutación

1- Se almacenan para el primer padre, todos los nodos que contienen símbolos no terminales (salvo el nodo raíz) en el conjunto NT.
NT={(A(1)),(C(2)),(D(3)),(S(1,1),(prB(3,1)),(B(3,2)),(psB(3,3)),(P(3,1,1)), (B(3,1,2)) ,(P(3,3,1))}

2- Se elige de manera aleatoria un nodo cualquiera del entre los que componen el conjunto NT

El nodo elegido es: Nodo de mutación = (psB (3,3)).

3- Se busca el nodo padre del nodo elegido anteriormente (D, (3)) y se almacenan en R todos los consecuentes de las reglas pertenecientes a la gramática donde "D" aparece como antecedente.

R = [prB B psB, prB B, B psB, B]

4- Se obtiene la derivación principal $D \rightarrow prB\ B\ psB$. La longitud de la derivación principal es, l = 3, y la posición que ocupa dentro del nodo de cruce dentro del consecuente de la derivación principal es p = 3. Por consiguiente, la tupla obtenida es la siguiente:

T = (3,3, prB B psB)

5- Se eliminan de R todos aquellos consecuentes cuya longitud es diferente de la longitud de la derivación principal.

R = [prB B psB]

6- Cada elemento de R, se compara con los símbolos del consecuente de la derivación principal excepto aquel que se encuentra en la posición del nodo de cruce.

R = [prB B psB]

7- Se calcula el conjunto X, formado por todos aquellos símbolos en los que los consecuentes de R, que están la misma posición, *p*, que el nodo de cruce. Por lo tanto, se elige para cada elemento de R, el símbolo ubicado en la tercera posición.

X = [psB]

8- Se toma aleatoriamente como símbolo elegido (SE) del conjunto X el símbolo psB, que en este caso es el único símbolo.

9- Se calcula la profundidad a la que se encuentra el nodo de mutación, y se efectúa la resta entre la profundidad máxima (D) y dicha profundidad, obteniendo así, la longitud de la mutación (LM).
LM = 5 - 3 = 2

10- Se etiqueta el símbolo elegido psB como el actual no terminal, y se asigna el valor 0 a la profundidad actual.

11- Almacenamos en el conjunto PC las producciones que cumplen: *PA + L (psB ::= α) ≤ 5*. Por tanto:
PC = {psB → S psS, psB→ P psP, psB → H psH, psB → S, psB → P, psB → H}

12- Se escoge de manera aleatoria la producción psB →H del conjunto PC

13- Por cada no terminal B ε α, anotamos B como el actual no terminal SE, por lo que anotamos H como el símbolo actual no terminal y repetimos los pasos 11, 12 y 13 para este símbolo, incrementando en una unidad el valor de la profundidad actual.

Con H:

11-Calculamos el contenido de PC = {H →h-gra, H →h-peq}.

12-Se escoge aleatoriamente la producción H →h-gra.

13-Como no existen símbolo no terminales B ε α, se continua por el paso 14

14-Se reemplaza el subárbol generado, cuya raíz es el nodo de mutación por el árbol generado en los pasos 11, 12 y 13.

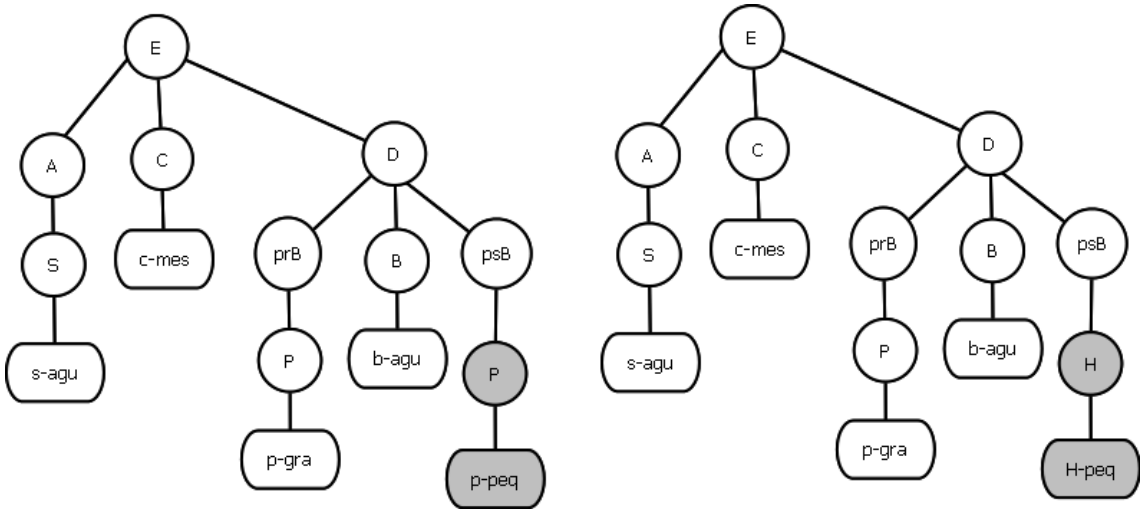

Figura 3-9: Ejemplo de mutación. A la izquierda el individuo inicial, y a la derecha el individuo producido por la mutación.

3.5 DISTANCIA MÍNIMA DE EDICIÓN SIMBÓLICA

En campos como por ejemplo, Teorías de la Información o Ciencias de la Computación, se denomina Distancia de Levenshtein, distancia mínima de edición o distancia entre palabras, al menor numero de operaciones para transformar una cadena de caracteres en otra. Por operación entendemos una inserción, eliminación o sustitución.

Tiene una gran utilidad en programas que determinan cuanto son de similares dos cadenas de caracteres, como pueda ser el caso de los correctores ortográficos.

Por ejemplo, la distancia mínima de edición entre “chile” y “silla” es de 3, por que al menos necesitamos cuatro ediciones elementales para cambiar uno en el otro.

1. casa → cala (sustitución de 's' por 'l').
2. cala → calla (inserción de 'l' entre 'l' y 'a').
3. calla → calle (sustitución de 'a' por 'e').

Esta considerada como una generalización de la distancia de Hamming.

4-PLANTEAMIENTO DEL PROBLEMA

4.1 MOTIVACION

Son numerosas las herramientas creadas y los estudios realizados sobre la programación genética que han sido desarrollados para abordar una amplia variedad de dominios y objetivos. La idea que pretende alcanzar este trabajo, es la de crear una herramienta que sirva para cualquier gramática libre de contexto y para cualquier población perteneciente a dicha gramática, ya que las herramientas creadas con anterioridad eran útiles en un único caso, y funcionaban bajo un gramática libre de contexto previamente establecida.

4.2 OBJETIVOS

El objetivo principal de este trabajo, es la creación de una herramienta que sea capaz de analizar una gramática libre de contexto, a partir ese punto, cree los arboles de los individuos de la población inicial y a partir de ahí, los distintos algoritmos de la programación genética guiada por gramáticas son aplicados para encontrar una solución a un determinado problema. A continuación se detallan cada uno de los requisitos que se han tenido en cuenta a la hora de desarrollar esta herramienta.

- *Req.1*-La herramienta tiene que ser capaz de analizar un fichero texto que contenga una gramática libre de contexto en un determinado formato

- *Req.2*- La herramienta será capaz analizar un fichero de texto, que contiene los individuos de la población inicial, para a partir de la gramática, crear los arboles de derivación de cada individuo.

- *Req.3*- Como la gramática puede ser diferente en cada caso, el usuario podrá introducir un fichero de código fuente en C#(.Net) mediante el cual se podrá evaluar el grado de adaptación al medio de los individuos que conforman cada generación.(*función fitness*)

- *Req.4*- Al ser la gramática diferente en cada caso, el usuario deberá introducir un fichero (.csv) para que la herramienta pueda calcular la distancia mínima de

edición ponderada entre los individuos de la misma generación, y también con los individuos que conforman la población inicial.

- *Req.5*- El usuario debe poder especificar la profundidad máxima que deben tener los individuos de la población. Esto es necesario, ya que a partir del fichero de individuos se crean los arboles de derivación de cada individuo y como máximo podrá tener ese valor introducido como profundidad máxima.

- *Req.6*-La herramienta tiene que permitir al usuario poder seleccionar un operador de selección entre los 4 propuestos:

    - *Req.6.1*- El operador de selección torneo tiene que permitir introducir al usuario el numero de individuos que participaran en cada torneo.

    - *Req.6.2*- El operador de selección ruleta tiene que permitir introducir al usuario el porcentaje de individuos que se seleccionaran para su posterior reproducción.

    - *Req.6.3*- El operador de truncamiento tiene que permitir al usuario introducir el umbral fitness a partir del cual los individuos que tengan un valor fitness mayor al umbral establecido queden seleccionados para la reproducción.

    - *Req.6.4*- El operador de selección generacional tiene que seleccionar el 100% de los individuos para su posterior reproducción.

- *Req.7*- La herramienta tiene que permitir al usuario poder seleccionar entre los dos operadores de cruce (WX o GBC) y devolver los descendientes correspondientes.

- *Req.8* - La herramienta tiene que permitir seleccionar el porcentaje de individuos a mutar en cada generación. En el caso de que este porcentaje, sea menor que el

porcentaje necesario para al menos mutar un individuo, dicho valor, se ira acumulando hasta poder al menos mutar un individuo en alguna de las generaciones.

- *Req.9*- La herramienta tendrá que aplicar el operador de reemplazamiento a la generación obtenida, con respecto a la población de la generación anterior sobre la que se ha aplicado la reproducción.

- *Req.10*- La herramienta proporcionara al usuario la posibilidad de seleccionar una, dos o tres condiciones de parada.

    - *Req.10.1*- La primera condición de parada indicara un número máximo de generaciones.

    - *Req.10.2*- La segunda condición evaluara el mejor valor fitness, por lo que hasta que un individuo no alcance o supere dicho valor, la herramienta ira obteniendo generaciones.

    - *Req.10.3*- La tercera condición de parada, tendrá dos parámetros. Uno de ellos será un porcentaje de individuos y el segundo un valor fitness. Esta condición funcionara de la siguiente manera, hasta que ese porcentaje de individuos indicados como parámetro no superen el valor fitness indicado, se irán obteniendo nuevas generaciones.

- *Req.11*-La herramienta proporcionara en la interfaz una visualización de los datos más relevantes a cada generación. Estos datos serán:
    - El mejor fitness
    - El peor Fitness
    - Media del fitness de la generación
    - Media de la distancia minimiza de edición simbólica ponderada con respecto a la generación actual.

  - Media de la distancia mínima de edición simbólica ponderada con respecto a la población inicial.
  - Desviación típica.

- *Req.12*- La herramienta proporcionara un fichero de resultados (.csv) mediante el cual, se podrán explorar de una forma mas detallada los resultados obtenidos en cada una de las generaciones.

- *Req.13*- Se podrá ejecutar en la opción "Paso a Paso", para lo cual, no hace falta introducir ninguna condición de parada ya que el usuario decide cuando dejar de producir nuevas generaciones.

- *Req.14* - Para acceder al fichero de resultados de una forma rápida, la herramienta ofrece un botón mediante el cual se podrá abrir el archivo generado de una manera rápida.

- *Req.15*- El usuario podrá especificar sobre que funcion fitness trabajar, ya sea por la distancia mínima de edición con respecto a la población actual o inicial, o tanto mediante la función de adaptación introducida por fichero. Esto engloba modificaciones en los algoritmos de selección, reemplazamiento y la condición de parada. Ya que si el usuario selecciona que quiere utilizar la función de adaptación por fichero, se seleccionarían los individuos con mayor valor fitness, pero sin embargo, el usuario quisiese trabajar con distancias simbólicas, se seleccionarían los individuos con valor mas bajo, ya que estamos buscando soluciones cercanas a la población inicial. Ocurriría lo mismo con el operador de reemplazamiento, y también en las distintas condiciones de parada.

5-RESOLUCIÓN DEL PROBLEMA

5.1 INTRODUCCION

A lo largo de este capitulo, se presentaran las diferentes soluciones propuestas para hacer frente al desarrollo de los requisitos mencionados en el capitulo previo. El primer punto abordara la construcción del algoritmo para crear el árbol de derivación de un individuo a partir de la gramática. En el segundo se explicara la forma de compilar el fichero de código fuente con la función fitness y en el tercer punto, se abordara la implementación del cálculo de la distancia simbólica de edición, con respecto a un fichero de pesos. El último punto, presentara la aplicación desarrollada para trabajar con sistemas de programación genética guiada por gramáticas.

5.2 DEFINICION DEL ALGORITMO DE GENERACION ALEATORIA DE INDIVIDUOS

El algoritmo de generación aleatoria de individuos tiene como objetivo establecer la población inicial de un sistema de Programación Genética Guiada por Gramáticas. La característica fundamental que presenta este algoritmo reside en que a la hora de generar los individuos, no se eligen las producciones que lo conforman de manera totalmente aleatoria, sino que se eligen producciones que aseguran en todo momento que el individuo pertenece al lenguaje generado por la gramática y que la profundidad de su árbol de derivación no va a superar una cota previamente establecida. Esto proporciona un ahorro computacional importante en términos de tiempo y memoria dado que ningún individuo es descartado a la hora de formar parte de la población inicial.

El algoritmo recibe dos parámetros: el *número de individuos* (N) que componen la población y la *profundidad máxima permitida* (D) de los árboles de derivación.

El proceso de generación aleatoria de la población consta de tres pasos:

1) Se anota la longitud para cada producción de la gramática. Para ello se tienen en cuenta las siguientes reglas:

- La longitud de un símbolo no terminal A es el mínimo de las longitudes de todas sus producciones, y se denota por L(A).
- La longitud de un símbolo terminal a es 0 porque no deriva a nada, denotándose por L(a) = 0.
- La longitud de una producción A::= α es el resultado de sumar uno al máximo de las longitudes de los símbolos que componen la parte derecha, y se denota por L(A::= α).
- Las producciones que generan sólo símbolos terminales tienen longitud 1, denotándose por L(A:=a) = 1, $\forall A \in \Sigma_N$ y $\forall a \in \Sigma^*_T$.

2) Se calcula la longitud del axioma (mínimo de las longitudes de todas aquellas producciones cuyo antecedente es el axioma). Esta longitud determina la *profundidad mínima permitida* (d) para un individuo válido de esa gramática.

3) Repetir N (número de individuos de la población) veces:

3.1) Se escoge aleatoriamente un valor comprendido entre la profundidad mínima permitida (d) y la profundidad máxima permitida (D). Este valor corresponde a la profundidad máxima elegida (p) para el presente individuo. El algoritmo garantiza la generación de un individuo cuya profundidad está comprendida entre d y p.

3.2) Se etiqueta el axioma como el actual no terminal A y se asigna el valor 0 a la profundidad actual (PA).

3.3) Se selecciona aleatoriamente una producción de la forma A ::= α, ($\alpha \in \{\Sigma_N \cup \Sigma_T\}^*$) que cumpla: PA + L(A ::= α) ≤ p.

3.4) Por cada no terminal B ∈ α, se anota B como el actual no terminal A y se repiten los pasos 3.3 y 3.4, incrementando en una unidad el valor de PA.

5.3 DEFINICION DEL ALGORITMO DE CONSTRUCCION DEL ARBOL DE DERIVACIÓN DE UN INDIVIDUO CON RESPECTO A UNA GRAMATICA

El algoritmo de generación del árbol de derivación dado un individuo perteneciente a una gramática, tiene como objetivo la creación de dicho árbol para posteriormente, aplicar los diferentes operadores genéticos guiados por gramáticas. Este algoritmo va realizando una búsqueda en profundidad de derecha a izquierda de acuerdo a un límite de profundidad establecido previamente. Esto es un proceso relativamente lento, ya que por ejemplo si tenemos una gramática ambigua con un número alto de producciones y un valor de profundidad alto, esta parte de la ejecución tarda algunos segundos.

Este algoritmo, tiene que tener una preparación inicial antes de poder definir el árbol de derivación de un individuo.

El primer proceso que requiere este algoritmo, es la lectura del fichero de gramática, y anotar las producciones de la misma los símbolos terminales y no terminales de la misma. Cada producción esta definida por tres atributos:

- El antecedente(string)
- El consecuente (ArrayList).

- Axioma (valor booleano que con el valor true, indica que ese Símbolo No Terminal correspondiente el axioma de la gramática).

Una vez, la gramática con la que queremos trabajar a sido definida, el proceso continua realizando el calculo de los distintos conjuntos FIRST(α) de los símbolos terminales.

Si α es cualquier cadena de símbolos gramaticales, se considera FIRST (α) como el conjunto de símbolos terminales que encabezan las cadenas derivadas de α. Si α = * => λ, entonces λ también está en FIRST (α).

Para calcular FIRST(X) para algún símbolo X de la gramática, se aplican las siguientes reglas hasta que no se pueda añadir nada nuevo al conjunto FIRST:

1. Si X es terminal, entonces FIRST(X) es {X}.

2. Si X es no terminal y existe la producción X → λ, entonces añadir FIRST(X).

3. Si X es no terminal y X → Y1 Y2.... Yk es una producción entonces, para todo *i* (con *i* variando desde 1 hasta k) tal que Y1, Y2,..., Yi-1 sean todos no terminales y FIRST (Y1), FIRST (Y2),..., FIRST (Yi-1) contengan todos λ, se añaden todos los símbolos no nulos de FIRST (Yi) a FIRST(X). Finalmente, si λ está en FIRST (Yj) para j = 1, 2,..., k (o sea, en todos), entonces se añade λ a FIRST(X). Dicho de otra forma, lo anterior significa que todos los elementos de FIRST (Y1), excepto λ, pertenecen también a FIRST(X). Si Y1 no deriva λ, entonces ya ha terminado el cálculo de FIRST(X), pero en caso contrario, es decir, si Y1= *=> λ, entonces todos los elementos de FIRST (Y2) excepto λ pertenecen también a FIRST(X), y así sucesivamente. Finalmente, si todos los Yi derivan λ, entonces λ se añade a FIRST(X).

Una vez los conjuntos FIRST (α), se pasa a construir el árbol de derivación de los individuos. Dicho algoritmo recibe el individuo (como un string) y un límite de profundidad establecido desde la interfaz del usuario.

Este algoritmo, mostrado en la tabla 5.1, es una de las partes fundamentales de la herramienta, ya que de su correcto funcionamiento, depende el resto de la ejecución.

```
Axioma = busquedaAxioma (Gramatica)
pilaSimbolos.Push(Axioma.getAntecedente())
pilaProfundidad.Push(0)
do{
        Si profundidad < profMaxima && ptrIndividuo < longitud individuo
                Si cabezaPila es Terminal
                        Si  cabezaPila == ptrIndividuo
                                pilaSimbolos.Pop()
                                pilaProfundidad.Pop()
                                ptrIndividuo++
                                guardaUltimoEstadoAceptado();
                        ELSE
                                leeUltimoEstadoAceptado();
                ELSE
                        Si ptrIndividuo pertenece al conjunto FIRST(cabezaPila)
                                pilaSimbolos.Pop()
                                pilaprofundidad.Pop()
                                producción = buscaProduccion()
                                parse.Add(produccion)
                                profundidad++
                                pilsaSimbolos.Push(producción.getConsecuente)
                                pilaProfundidad.Push(profundidad)
                                        (Tantos como la longitud de consecuente)
                                guardaUltimoestadoActeptado();
                        ELSE
                                leeUltimoEstadoAceptado();
        ELSE
                leeUltimoEstadoAceptado();
}
while(pilaSimbolos = Vacia && pilaProfundidad = Vacia)
```

Taba 5-1: Algoritmo de creación del árbol de derivación de un individuo, a partir de una gramática previamente definida.

5.4 COMPILACION EN TIEMPO DE EJECUCION DE UN FICHERO DE CODIGO FUENTE EN C# (.NET)

El desarrollo de esta parte de la herramienta, es necesaria para poder ejecutar la función fitness introducida por el usuario mediante un fichero de código fuente. Gracias a la librería "System.CodeDom" del lenguaje C#, podemos compilar código y gracias a la reflexión, podemos ejecutarlo.

El uso que se le puede dar es prácticamente ilimitado, aunque no es fácil de comprender, ya que lo primero que hay que decidir, es para que lo queremos utilizar

exactamente. En el caso de este proyecto, esto es necesario, ya que en ocasiones tendremos que compilar ficheros de C# en tiempo de ejecución (cuando el usuario seleccione que la aplicación trabaje con un fichero de código fuente como función fitness). En nuestro caso, todos los ficheros tendrán una parte en común y bien definida, ya que la función fitness por fichero siempre recibirá el mismo argumento (el string del individuo) y devolverá siempre un mismo tipo de valor numérico.

```
private object CreaYCompilaHolaMundoClass() {
            CompilerParameters cp = new CompilerParameters();
            cp.ReferencedAssemblies.Add("system.dll");
            cp.GenerateExecutable = false;
            cp.GenerateInMemory = true;
            StringBuilder code = new StringBuilder();
            code.Append("using System; \n");
            code.Append("using System.Windows.Forms; \n");
            code.Append("using System.Data; \n");
            code.Append("using System.Xml; \n");
            code.Append("using System.Reflection; \n");
            code.Append("namespace Prueba.Compilador { \n");
            code.AppendFormat("  public class {0} ", "HolaMundo");
            code.Append("{ ");
            //Clases de apoyo
            //GetValue a string
            code.Append(" public void HazTuTrabajo() \n");
            code.Append("  {");
            code.Append(" MessageBox.Show(\"Hola mundo\");");
            code.Append(" }\n");
            code.Append("} }");
            CompilerResults cr =
CodeDomProvider.CreateProvider("C#").CompileAssemblyFromSource(cp, code.ToString());
            if (cr.Errors.HasErrors)
            {
                StringBuilder error = new StringBuilder();
                error.Append("Error al compilar: ");
                foreach (CompilerError err in cr.Errors)
                {
                    error.AppendFormat("{0}\n", err.ErrorText);
                }
                throw new Exception("Error al compilar: " + error.ToString());
            }
            Assembly a = cr.CompiledAssembly;
            return a.CreateInstance("Prueba.Compilador.HolaMundo");
        }
```

Taba 5-2: Ejemplo de compilación y ejecución de un fichero con código fuente(C#).

En el ejemplo de la tabla 5-2 podemos ver como compilar un fichero de código fuente en tiempo de ejecución. Para esta la herramienta desarrollada, la única modificación

reseñable con respecto a lo anterior, es a la hora de generar el "StringBuilder", que se genera leyendo el fichero de código fuente línea a línea y añadiendo dichas líneas al StringBuilder.

5.5 DISTANCIA MÍNIMA DE EDICION SIMBOLICA PONDERADA

El objetivo fundamental del cálculo de la distancia mínima de edición ponderada, tiene como el objetivo el cálculo de la distancia entre dos individuos, en función de unos valores llamados pesos, que es una de las entradas de la aplicación.

Este algoritmo es de tipo bottom-up, bastante común en programación dinámica. Esta apoyado en la utilización de una matriz (n+1) × (m +1), donde n y m, son las longitudes de los dos individuos. A continuación, se puede ver en pseudocódigo original de la función de LevenhteinDistance, que recibe como argumentos dos cadenas de distinta longitud y calcula la distancia mínima de edición entre ellos:

```
int DistanciaDeLevenhtein (char str1[1..lonStr1], char  str2[1..lonStr2])
        //d es la tabla con longitud de longStr1+1 filas y longStr2+1 columnas
        declare int d[0..lenStr1, 0..lenStr2]
        // i y j son declarados para iterar sobre la matriz
        declare int i, j, cost
        for i from 0 to lenStr1
                d[i, 0] := i
        for j from 0 to lenStr2
        d[0, j] := j
        for i from 1 to lenStr1
                for j from 1 to lenStr2
                        if str1[i] = str2[j] then cost := 0
                        else cost := 1
                        d[i, j] := minimum(
                                d[i-1, j] + 1,     // eliminación
                                d[i, j-1] + 1,     // inserción
                                d[i-1, j-1] + cost   // sustitución

                                )

        return d[lonStr1, lonStr2]
```

Taba 5-3: Algoritmo en pseudocódigo de la Distancia Mínima de Edición.

Para el desarrollo de esta herramienta, este algoritmo ha sido modificado para adaptarlo a lo que se necesitaba. La solución utilizada en la herramienta realiza los siguientes pasos:

1. Se realiza una lectura de un fichero .csv, mediante el cual el usuario, especifica mediante dos tablas unidimensionales (tabla de eliminación y sustitución) y una matriz (sustitución), el coste que tiene cada una de las posibles acciones.

2. A continuación realiza lo mismo que en la tabla 5-3, excepto cuando calcula el mínimo de eliminación (coste 1 en el algoritmo), inserción (coste 1 en el algoritmo) y el de sustitución (coste 1 en el algoritmo), que en nuestra herramienta, esos valores son leídos de las tablas unidimensionales de inserción y eliminación o de la matriz de sustitución.

5.6 SOFTWARE DESARROLLADO

5.6.1 PRESENTACION DE LA HERRAMIENTA

Se ha desarrollado una aplicación que implementa un sistema completo de Programación Genética Guiada por Gramáticas. Esta implementación, ha sido implementada en C# (.Net). Las funcionalidades más importantes de es aplicación son las siguientes:

- Utilización de ficheros que recogen la definición de las gramáticas libres de contexto, los individuos que conforman la población inicial, los pesos a utilizar en el calculo de la distancia mínima de edición y opcionalmente el fichero de código fuente (C#), que calcule el grado de adaptación de cada individuo.

- Posibilidad de elegir entre 3 métodos de cálculo de fitness, según la distancia mínima de edición entre los individuos de la misma población, con respecto a la población inicial u otra función cualquiera que el usuario haya programado con anterioridad.

- Incorporación de los métodos de selección mas utilizados:
    - Torneo.
    - Ruleta.
    - Truncamiento.
    - Generacional.

- Utilización de los métodos de cruce que generan individuos validos para la gramática libre de contexto:
    - Whigham.
    - Grammar Based Crossover.

- Utilización del método de mutación GBM (*Grammar Based Mutation*).

- Probabilidad de ocurrencia de mutaciones.

- Utilización del método de reemplazamiento según el tipo de valores que el usuario haya definido como función fitness.

- Control de profundidad de los individuos.

- Definición de varias condiciones de parada:
    - Numero de generaciones.
    - Mejor valor fitness.
    - N individuos $\geq \leq$ valor (*Depende del que parámetro que queramos que actúe como valor fitness*).

- Posibilidad de que el usuario pueda ir generación a generación, sin especificar ninguna condición de parada.

- Creación de fichero de resultados, médiate el cual, se pueden ver los diferentes valeros de las diferentes funciones fitness y estadísticas relacionadas.

5.6.2 PRESENTACION DE LA INTERFAZ GRAFICA Y FUNCIONALIDAD DE LA APLICACIÓN

La herramienta, presenta una interfaz grafica muy sencilla e intuitiva, en la que se permite la parametrización y control de todos los parámetros del sistema de Programación Genética Guiada por Gramáticas.

Figura 5-1: Pantalla principal de la herramienta G3ET

A continuación, se indica una breve descripción de cada uno de los campos que conforman la herramienta:

- Fichero de Gramática: Indica la ruta del fichero de gramática libre e contexto (.txt), el cual es utilizado para resolver el problema.

- Generar Individuos aleatoriamente: Permite la creación de los individuos aleatoriamente, seleccionado el número de individuos a crear.

- Fichero de Individuos: Indica la ruta del fichero (.txt) de los individuos que conforman la población inicial.

- Fichero de pesos: Indica la ruta del fichero (.csv) de los pesos que serán utilizados para realizar los cálculos de la distancia mínima de edición entre los individuos.

- Fichero función fitness: Indica la ruta del fichero de código fuente(C#), que será utilizado como medida de grado de adaptación de los individuos generados en cada una de las generaciones.

- Control de profundidad: Indica el limite de profundidad que podrán tener cada uno de los arboles de derivación de cada individuo.

- ComboBox que permite seleccionar el operador de selección. En función del operador seleccionado se habilitaran uno de los siguientes textbox:
    - Torneo : Indica el numero de individuos que participaran en cada torneo
    - Ruleta: Indica el porcentaje de individuos que serán seleccionados aleatoriamente
    - Truncamiento: Indica el valor fitness mínimo que tendrá que tener un individuo para ser seleccionado.

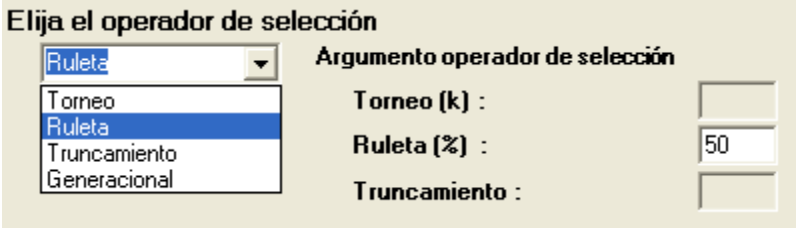

Figura 5-2: Imagen de ejemplo de selección de uno de los operadores de selección.

- ComboBox que permite seleccionar el operador de cruce.

- Porcentaje de individuos a mutar: Mediante el cual, se indica el numero de individuos que sufrirán una mutación en cada iteración.

- Numero Máximo de Generaciones: Indica el numero de generaciones que se generaran a lo largo del proceso.

- Mejor Fitness: Especifica que cuando uno de los individuos de una generación alcance o supere ese valor, la herramienta detectara que ha llegado a su fin.

- Mejor Fitness según el número de individuos: Es idéntico al caso anterior, pero tenemos un campo habilitado para introducir el número de individuos que queremos que superen ese valor.

- Botón Paso a Paso: Mediante este botón se puede ir realizando la generación de generaciones de una en una y detenerse cuando el usuario considere oportuno.

- Botón Ejecutar: Ejecuta el proceso desde el principio, hasta que se cumple una de las condiciones de parada.

- Abrir Resultados: Es utilizado para abrir de una forma rápida y cómoda el fichero de resultados.

- También comentar, que en la parte derecha de la aplicación, se harán un volcado de los datos más relevantes en cada generación, que el usuario podrá leer de una forma rápida.

5.6.3 FICHEROS DE ENTRADA DE LA HERRAMIENTA

Como hemos podido observar a lo largo de lo comentado en los capítulos, esta herramienta, recibe 4 ficheros como entrada. A continuación se especificara el formato correcto de cada uno de ellos:

- Fichero de Gramática (.txt): Recoge la gramática libre de contexto que representa el problema a resolver. El formato correcto de este fichero lo podemos observar en la siguiente figura:

```
Terminales = { a c d }
NoTerminales = { A C D }
Axioma = E
Producciones = {
        E -> A C D
        A -> a A a
        A -> a
        D -> d D d
        D -> d D
        D -> d
        C -> C c
        C -> c C
        C -> c
        }
```

Tabla 5-4: Ejemplo de formato de Fichero de gramática

- Fichero de Individuos (.txt): Del cual se obtienen los individuos que conforman la población inicial, estando cada uno en una línea diferente.

```
s-sua p-gra s-sua c-agu b-agu h-peq b-agu
s-agu c-irr h-gra b-sua h-gra b-sua
s-agu c-mes b-agu
p-gra b-agu h-gra s-agu c-agu p-gra b-agu p-gra
b-agu p-gra b-agu h-gra s-agu c-agu p-gra b-agu p-gra
s-agu b-agu c-agu p-gra b-agu p-gra
```

Tabla 5-5: Ejemplo de formato de Fichero de individuos

- Fichero función Fitness (.cs): Mediante el cual se puede obtener la función de evaluación (fitness).

```
susing System;
namespace Fitness
            {
            class Fitness
              {
                 static public int Main ( string str )
                  {
                       return str.Length;
                  }
              }
            };
```

Tabla 5-4: Ejemplo de formato de Fichero de función fitness

- Fichero de pesos (.csv): Especifica el valor de cada una de las acciones de inserción, eliminación o sustitución a la hora de calcula la distancia mínima de edición.

	A	B	C	D	E	F	G	H
1	Recuerde que los simbolos deben ir en el mismo orden en el que estan especificados en la gramatica							
2	Insercion							
3	a	1						
4	c	2						
5	d	4						
6	Eliminacion							
7	a	1						
8	c	2						
9	d	4						
10	Sustitucion							
11		a	c	d				
12	a	0	1	3				
13	c	1	0	4				
14	d	3	4	0				

Figura 5-3: Ejemplo de formato de Fichero e pesos desde Microsoft Excel.

6-RESULTADOS Y ANALISIS

6.1 INTRODUCCIÓN

Este capitulo está centrado en realizar un estudio de los resultados en función de los algoritmos genéticos y técnicas de evaluación dentro del marco de la Programación Genética Guiada por Gramáticas. En la primera parte del capitulo, se expondrá la población de individuos inicial, la gramática libre de contexto sobre la que se realizaran los experimentos y el fichero de pesos utilizado para el calculo de la distancia mínima de edición. En la segunda parte, se explicara cada uno de los resultados obtenidos de la experimentación realizada sobre estos ficheros de entrada.

6.2 FICHEROS DE ENTRADA

Para la realización de los experimentos, se utilizaran siempre los mismos ficheros de entrada. El fichero de individuos, contendrá una población inicial de 16 individuos, generados por la gramática definida en la tabla [3-1]. A continuación, se puede observar el contenido del fichero de individuos que conforman la población inicial, formados por la gramática libre de contexto definida en el capitulo 3.

```
s-sua p-gra s-sua c-agu b-agu h-peq b-agu
s-agu c-irr h-gra b-sua h-gra b-sua
s-agu c-mes b-agu
p-gra b-agu h-gra s-agu c-agu p-gra b-agu p-gra
b-agu p-gra b-agu h-gra s-agu c-agu p-gra b-agu p-gra
s-agu b-agu c-agu p-gra b-agu p-gra
s-agu p-gra c-mes b-agu
s-agu c-mes b-agu h-peq
s-sua p-gra s-sua c-agu b-agu s-sua h-peq b-agu
s-sua p-gra h-gra p-gra s-sua p-peq c-agu b-agu h-peq b-agu
s-sua p-gra h-peq s-sua c-mes b-agu h-peq b-agu
b-agu p-peq s-sua p-gra s-sua c-agu b-agu h-peq b-agu
s-sua p-gra s-sua c-agu b-agu h-peq p-gra b-agu
s-sua p-gra s-sua c-agu b-agu h-peq h-peq p-peq b-agu
s-sua h-gra p-gra s-sua c-mes b-agu h-peq b-agu
s-sua p-gra b-agu s-sua c-agu b-agu h-peq b-agu s-sua p-gra
```

Tabla 6-1: Individuos que conforman la población inicial generados por la gramática especificada en la tabla [3-1.]

En las siguientes tablas, se puede observar el contenido de las tablas y matrices de pesos que se utilizaran para realizar el calculo de la distancia mínima de edición entre los individuos que conformen las distintas generaciones.

Inserción/Eliminación	
c-agu	1
c-mes	2
c-irr	4
s-agu	5
s-sua	5
b-agu	8
b-sua	9
p-gra	1
p-peq	3
h-gra	7
h-peq	2

Tabla 6-1: Tabla de pesos de inserción y eliminación

Sustitución											
	c-agu	**c-mes**	**c-irr**	**s-agu**	**s-sua**	**b-agu**	**b-sua**	**p-gra**	**p-peq**	**h-gra**	**h-peq**
c-agu	0	3	4	5	10	6	7	8	9	10	11
c-mes	2	0	4	5	10	6	7	8	9	10	11
c-irr	2	3	0	5	10	6	7	8	9	10	11
s-agu	2	3	4	0	10	6	7	8	9	10	11
s-sua	2	3	4	5	0	6	7	8	9	10	11
b-agu	2	3	4	5	10	0	7	8	9	10	11
b-sua	2	3	4	5	10	6	0	8	9	10	11
p-gra	2	3	4	5	10	6	7	0	9	10	11
p-peq	2	3	4	5	10	6	7	8	0	10	11
h-gra	2	3	4	5	10	6	7	8	9	0	11
h-peq	2	3	4	5	10	6	7	8	9	10	0

Tabla 6-2: Matriz de pesos de sustitución

6.3 CASOS DE EXPERIMENTACIÓN

6.3.1 VELOCIDAD DE CONVERGENCIA DE LA POBLACION SEGÚN LA DISTANCIA MINIMA DE EDICION ENTRE LOS INDIVIDUOS DE LA MISMA GENERACION

A lo largo de este punto, se expondrán una serie de resultados obtenidos a partir de los ficheros de entrada del capitulo anterior. Mediante estos experimentos, queremos mostrar la velocidad de convergencia de la población inicial hacia una solución, y realizar una comparación entre los distintos operadores.

6.3.1.1 CASO 1

Este caso tiene como objetivo estudiar la velocidad de convergencia para las siguientes entradas de la aplicación G3ET:

- Profundidad Máxima: 10
- Operador de Selección: Torneo (k = 3).
- Operador de Cruce: Whigham
- Porcentaje de Individuos a mutar en cada generación: 50%
- Condición de parada:
    - Número Máximo de Generaciones: 20(Se establece un número alto de generaciones, para poder estudiar mejor la velocidad de convergencia).

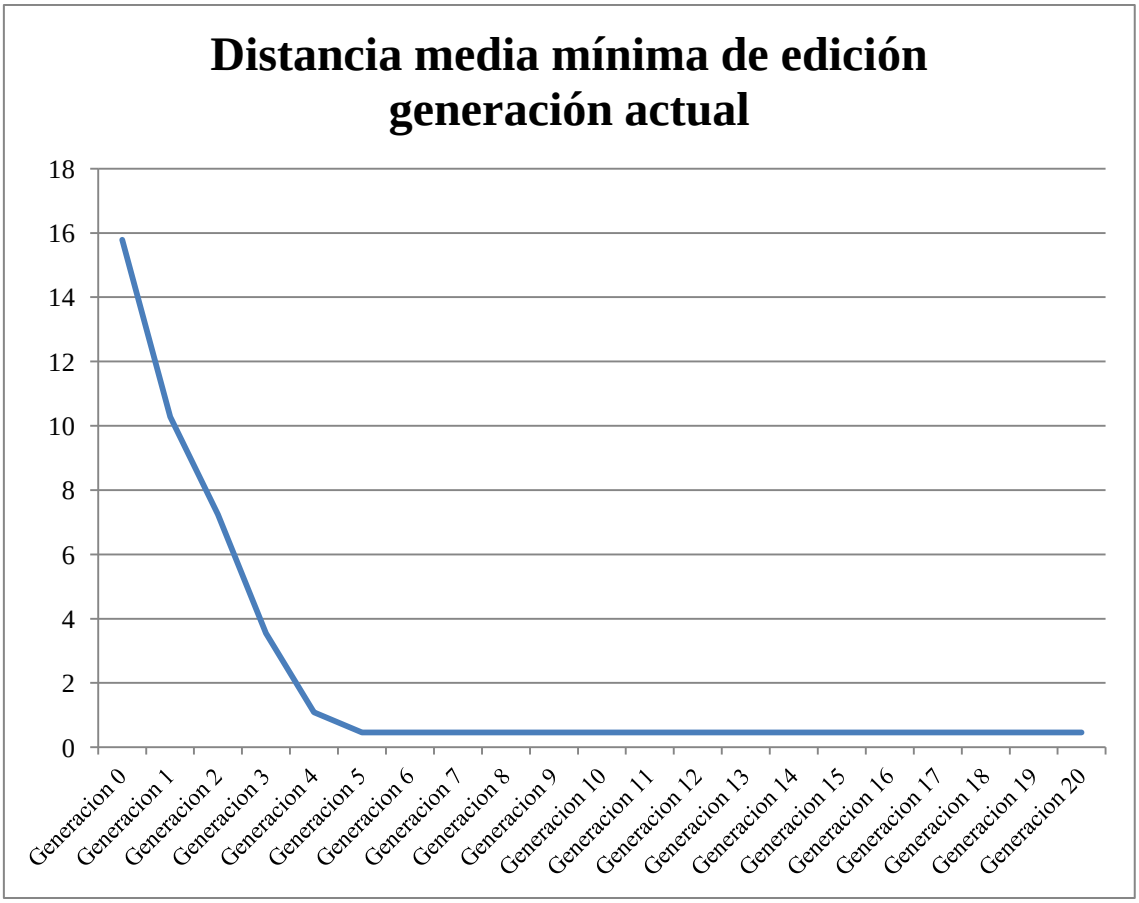

Figura 6-1: Velocidad de convergencia para el caso planteado

6.3.1.2 CASO 2

Este caso tiene como objetivo estudiar la velocidad de convergencia para las siguientes entradas de la aplicación G3ET:

- Profundidad Máxima: 10
- Operador de Selección: Ruleta(50% individuos)
- Operador de Cruce: Whigham
- Porcentaje de Individuos a mutar en cada generación: 50%
- Condición de parada:
    - Número Máximo de Generaciones: 20(Se establece un número alto de generaciones, para poder estudiar mejor la velocidad de convergencia).

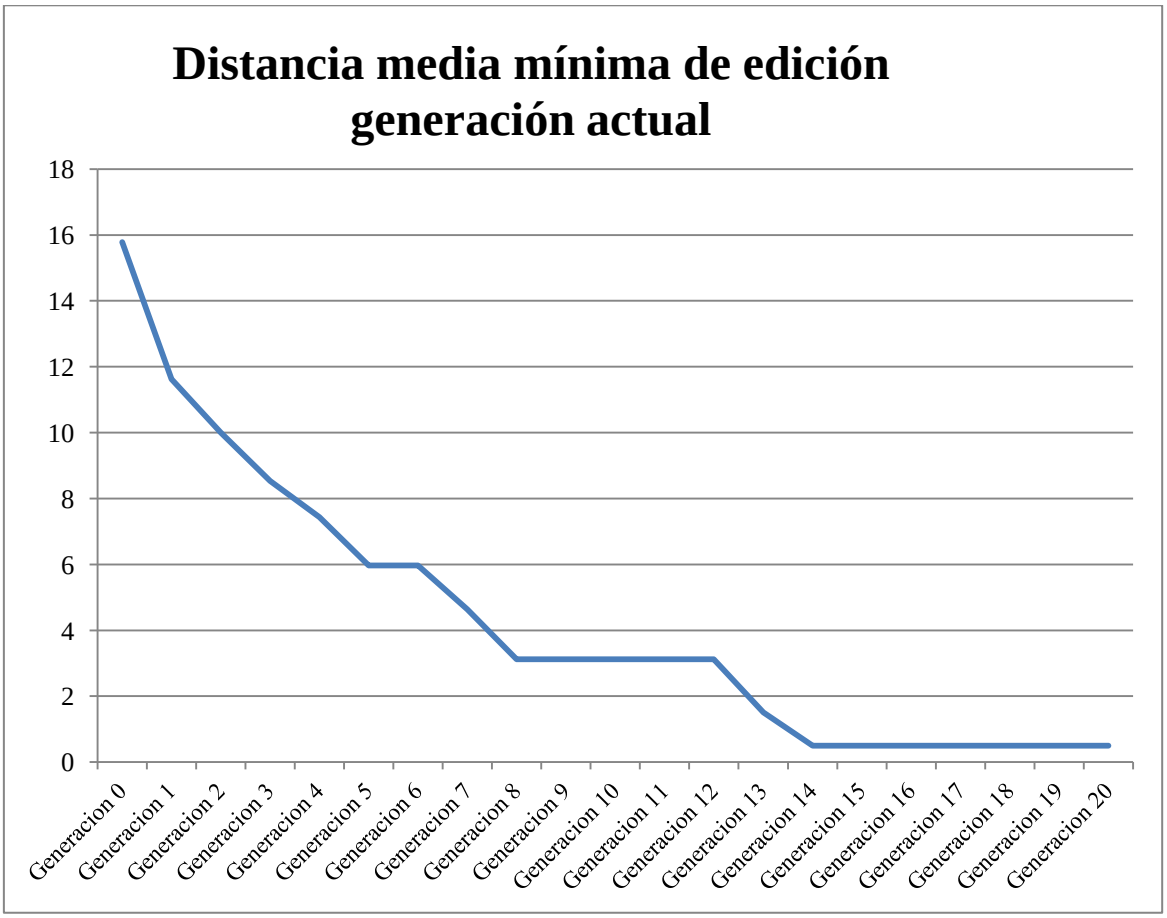

Figura 6-2: Velocidad de convergencia para el caso planteado

6.3.1.3 CASO 3

Este caso tiene como objetivo estudiar la velocidad de convergencia para las siguientes entradas de la aplicación G3ET:

- Profundidad Máxima: 10
- Operador de Selección: Truncamiento(valor 14)
- Operador de Cruce: Whigham
- Porcentaje de Individuos a mutar en cada generación: 50%
- Condición de parada:
    - Número Máximo de Generaciones: 20(Se establece un número alto de generaciones, para poder estudiar mejor la velocidad de convergencia).

Figura 6-3: Velocidad de convergencia para el caso planteado

6.3.1.4CASO 4

Este caso tiene como objetivo estudiar la velocidad de convergencia para las siguientes entradas de la aplicación G3ET:

- Profundidad Máxima: 10
- Operador de Selección: Generacional
- Operador de Cruce: Whigham
- Porcentaje de Individuos a mutar en cada generación: 50%
- Condición de parada:
    - Número Máximo de Generaciones: 20(Se establece un número alto de generaciones, para poder estudiar mejor la velocidad de convergencia).

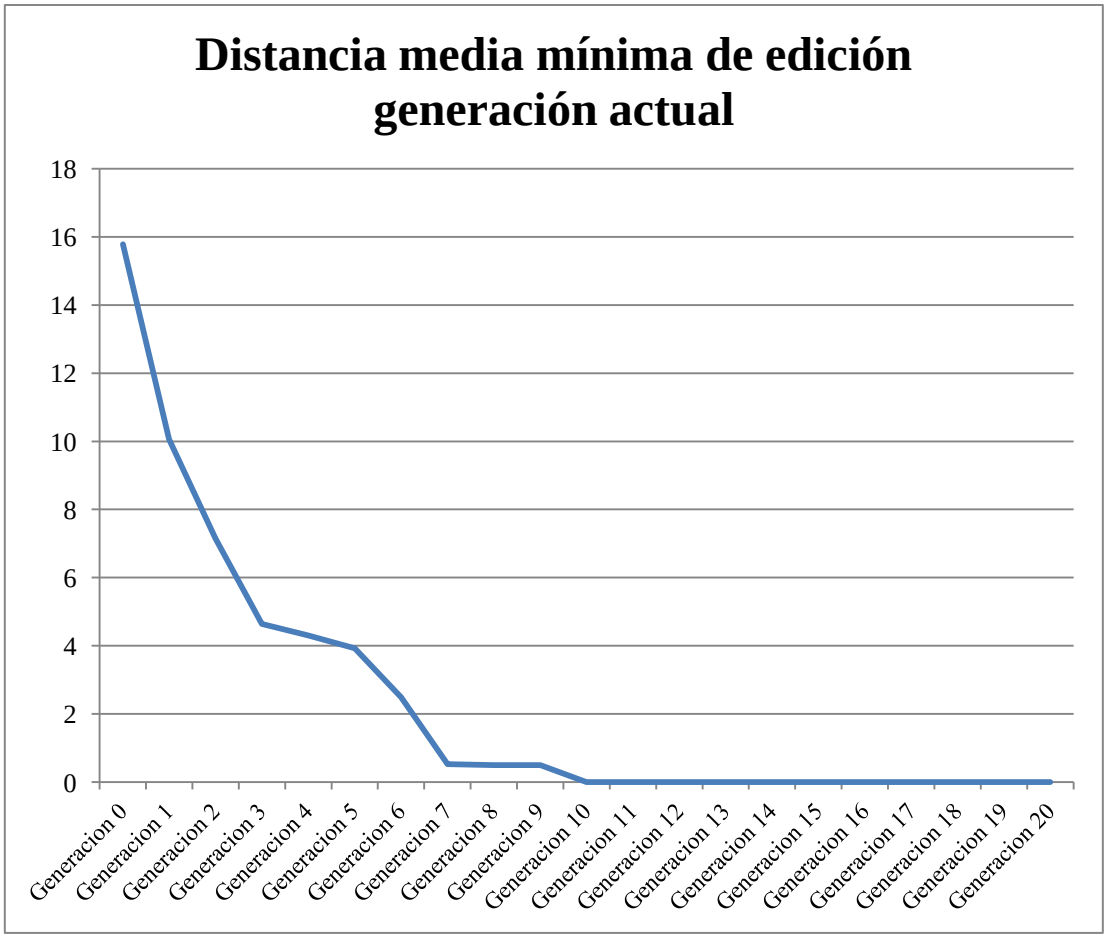

Figura 6-4: Velocidad de convergencia para el caso planteado

6.3.1.5 CASO 5

Este caso tiene como objetivo estudiar la velocidad de convergencia para las siguientes entradas de la aplicación G3ET:

- Profundidad Máxima: 10
- Operador de Selección: Torneo (k = 3).
- Operador de Cruce: GBC
- Porcentaje de Individuos a mutar en cada generación: 50%
- Condición de parada:
    - Número Máximo de Generaciones: 20(Se establece un número alto de generaciones, para poder estudiar mejor la velocidad de convergencia).

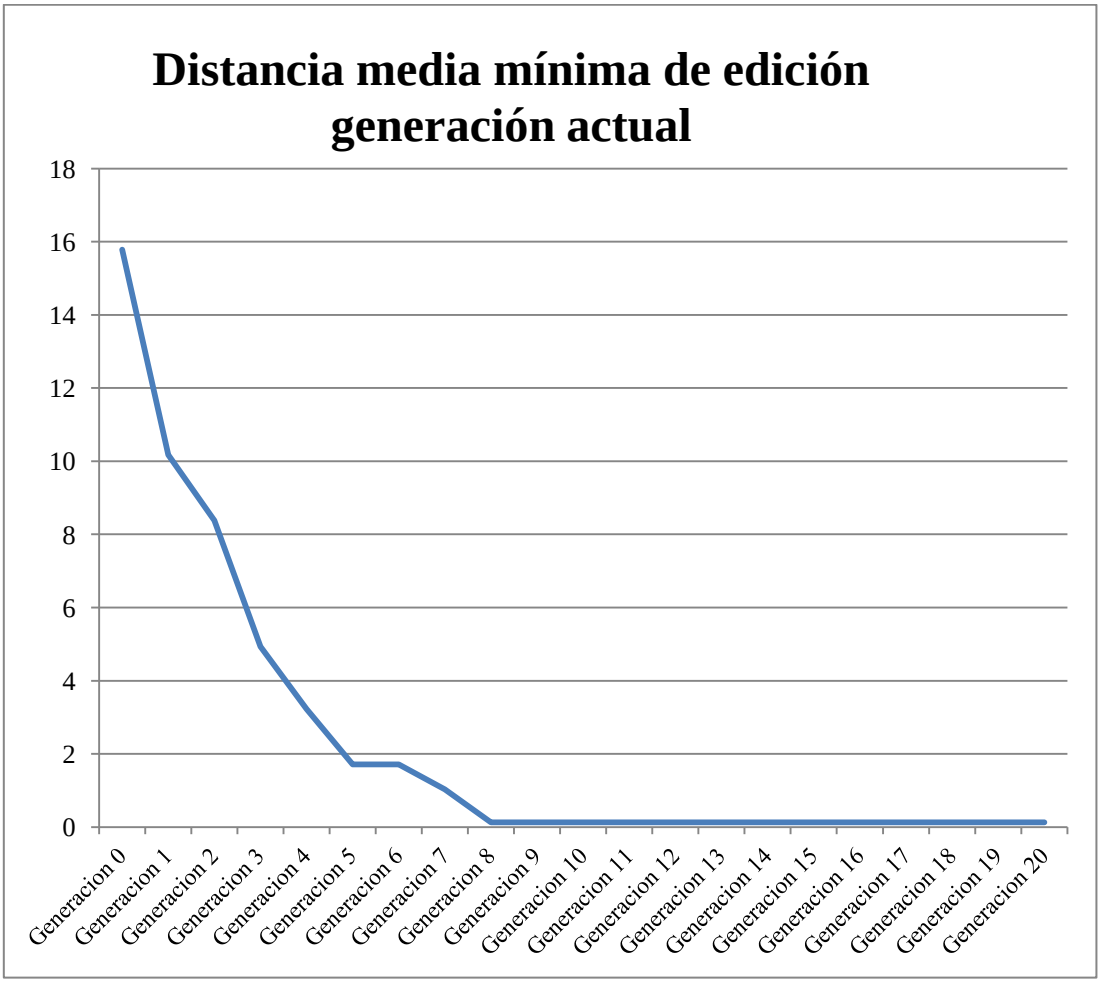

Figura 6-5: Velocidad de convergencia para el caso planteado

6.3.1.6 CASO 6

Este caso tiene como objetivo estudiar la velocidad de convergencia para las siguientes entradas de la aplicación G3ET:

- Profundidad Máxima: 10
- Operador de Selección: Ruleta(50% individuos)
- Operador de Cruce: GBC
- Porcentaje de Individuos a mutar en cada generación: 50%
- Condición de parada:
    - Número Máximo de Generaciones: 20(Se establece un número alto de generaciones, para poder estudiar mejor la velocidad de convergencia).

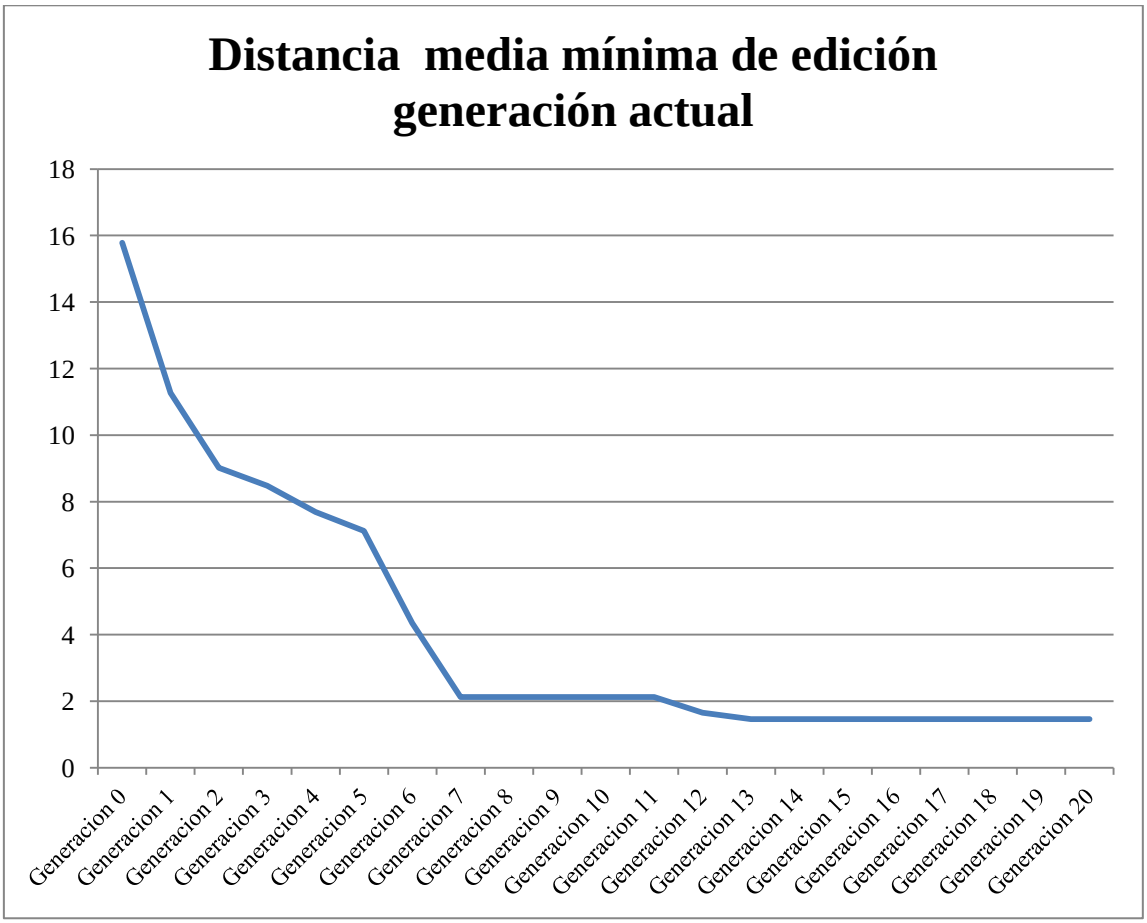

Figura 6-6: Velocidad de convergencia para el caso planteado

6.3.1.7CASO 7

Este caso tiene como objetivo estudiar la velocidad de convergencia para las siguientes entradas de la aplicación G3ET:

- Profundidad Máxima: 10
- Operador de Selección: Truncamiento(valor 14)
- Operador de Cruce: GBC
- Porcentaje de Individuos a mutar en cada generación: 50%
- Condición de parada:
    - Número Máximo de Generaciones: 20(Se establece un número alto de generaciones, para poder estudiar mejor la velocidad de convergencia).

Figura 6-7: Velocidad de convergencia para el caso planteado

6.3.1.8 CASO 8

Este caso tiene como objetivo estudiar la velocidad de convergencia para las siguientes entradas de la aplicación G3ET:

- Profundidad Máxima: 10
- Operador de Selección: Generacional
- Operador de Cruce: GBC
- Porcentaje de Individuos a mutar en cada generación: 50%
- Condición de parada:
    - Número Máximo de Generaciones: 20(Se establece un número alto de generaciones, para poder estudiar mejor la velocidad de convergencia).

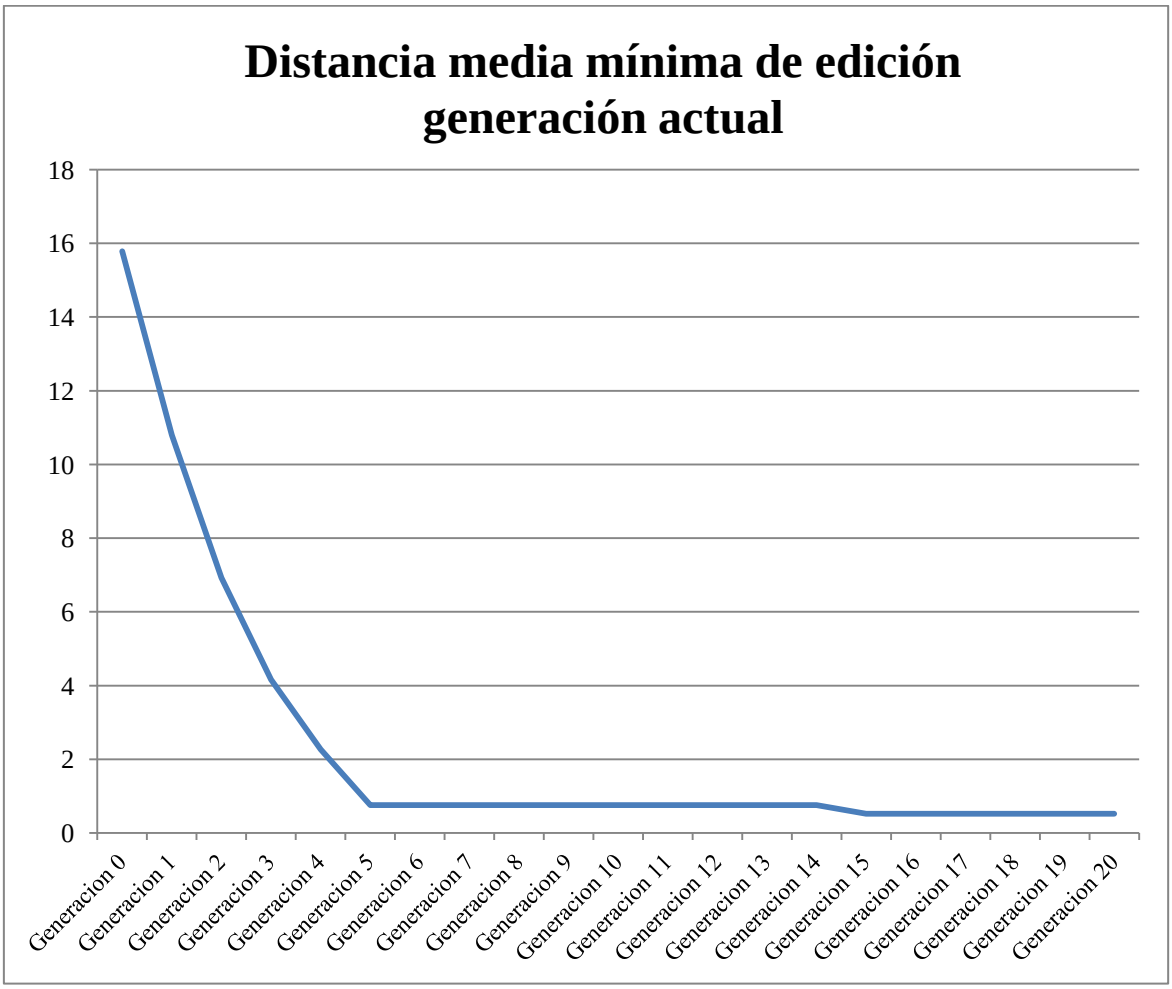

Figura 6-8: Velocidad de convergencia para el caso planteado

6.4 COMPARACION DE RESULTADOS

A la vista de los resultados obtenidos para los diferentes operadores de selección y cruce, se puede apreciar como el caso 6(*Op. Selección Ruleta y Op. Cruce GBC*) es el que peor velocidad de convergencia proporciona, debido a que al seleccionar los individuos de forma aleatoria es como una lotería. Este suceso, también es apreciable en el caso 2(*Op. Selección Ruleta y Op. Cruce WX*), que como se puede observar en el grafico inferior, es el que mas generaciones tarda en estabilizar sus valores.

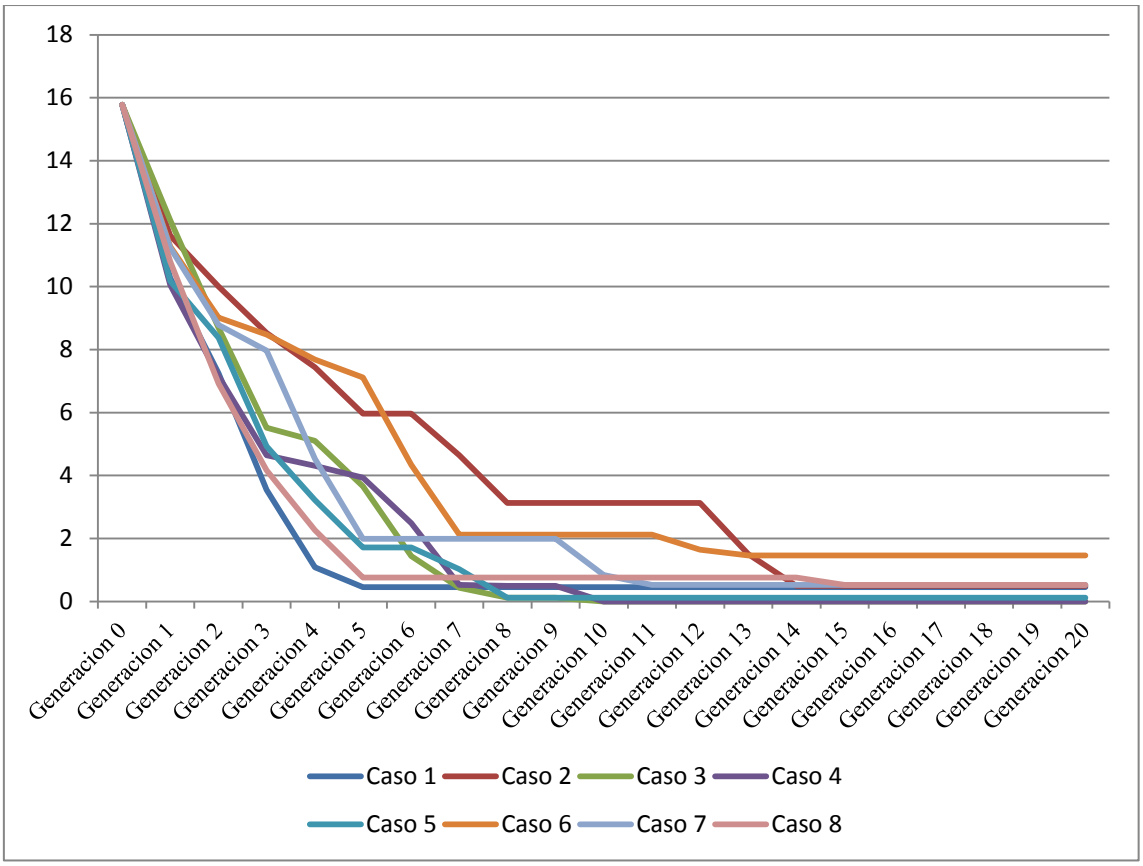

Figura 6-9: Grafica comparativa de los resultados obtenidos en los casos de experimentación anteriores.

Los mejores resultados, son obtenidos en el caso 3(*Op. Selección Truncamiento y Op Cruce WX*) y caso 4(*Op. Selección Generacional y Op Cruce WX*), ya que tienen una velocidad de convergencia muy alta, y llegan a generar una población homogénea, en la que todos los individuos son iguales.

El resto de casos, muestran velocidades de convergencia muy parecidos, no llegando a generar una población totalmente homogénea, pero si muy similar. Todo esto es debido a la función fitness utilizada en los experimentos, mediante la cual, se busca que la población a generar tenga la mínima distancia de edición simbólica entre los elementos de la misma población.

7-CONCLUSIONES

La realización de este trabajo fin de grado consiste en la implementación de un sistema automático para la búsqueda de soluciones basado en algoritmos de Programación Genética Guiada por Gramáticas. El problema a solucionar se representa mediante una gramática libre de contexto, utilizada para la generación de los individuos que conforman la población del sistema de G3ET.

En la herramienta desarrollada, todos los parámetros que conforman los distintos algoritmos de la programación genética, se pueden parametrizar, lo que le da una gran utilidad a la hora de buscar una solución sobre cualquier tipo de gramática libre de contexto aplicada a cualquier población inicial de individuos pertenecientes a dicha gramática.

Por todo lo expuesto a lo largo del presente trabajo fin de grado, esta herramienta, se convierte en una herramienta muy potente por la posibilidad de ser adaptada a cualquier tipo de problema cuya codificación de la solución pueda ser definida a través de una gramática libre de contexto.

El uso de la programación genética guiada por gramáticas frente a otras técnicas presenta ciertas ventajas, ya que permite la búsqueda de soluciones cuyas dimensiones no son conocidas previamente, ofreciendo un coste computacional relativamente pequeño. De esta forma, el tiempo de ejecución del sistema haciendo viables búsquedas de soluciones en espacios de búsqueda de gran tamaño se ve considerablemente reducido.

8-BILBIOGRAFIA

[1] Santamaría Falcón, Agustín: "Modelo de descubrimiento de conocimiento para series temporales numéricas aplicando métodos simbólicos". Tesis Doctoral, Madrid, 2011.

[2] Márquez, Fernando: "Programación genética guiada por gramáticas". Tesis Doctoral, Madrid, 2005.

[3] Koza, J. R., "Genetic Programming: On the Programming of Computers by Means of Natural Selection", MIT Press, Cambridge, MA, 1992.

[4] Baker J. E: "Reducing Bias and Inefficiency in the Selection Algorithm", Proceedings of the 1st International Conference on Genetic Algorithms, pp. 101-111,1987

[5] Brindle, A. , "Genetic Algorithms for Function Optimization", Tesis, University of Alberta, 1991.

[6] Crow, J. F. y Kimura, M.: "An Introduction to Populations Genetics Theory", New York Harper and Row, 1970.

[7] De Jong, K. A. : "Analysis of Behaviour of a class of Genetic Adaptive Systems". Tesis, University of Michigan, 1975.

[8] D'haeseleer, P.:"Context Preserving Crossover in Genetic Programming", Proceedings of the 1994 IEEE World Congress on Computational Intelligence,(1), Orlando, Florida, USA, 1994.

[9] Crawford-Marks R. y Spector, L.:"Size control via Size Fair Genetic Operators for grammar-guided genetic programming". Soft Computing – A Fusion of Foundations, Methodologies and Applications, Volume 11, Issue 10, pp. 943 – 955, 2007.

[10] Holland, J. H.: "Adaptation in Natural and Artificial Systems", University of Michigan, Ann Arbor, 1975.

[11] Whigham, P. A.: "Grammatically-Based Genetic Programming" Proccedings of the Workshop on Genetic Programming: From Theory to Real-World Applications, California, USA, pp- 33-41, 1995.

[12] Couchet, J., Manrique, D. y Rodriguez-Patón, A.:"Crossover and mutation operators form grammar-guided genetic programming", Soft Computing – A Fusion Foundations, Methodologies and Applications, Volume 11, Issue 10, pp 943-955. 2007

[13] Colaboradores de Wikipedia. *Computación evolutiva* [en línea]. Wikipedia, La enciclopedia libre, Disponible en: <http://es.wikipedia.org/w/index.php?title=Computacion_evolutiva&oldid=21389201>.

[14] Colaboradores de Wikipedia. *Programación evolutiva* [en línea]. Wikipedia, La enciclopedia libre, 2008 Disponible en: <http://es.wikipedia.org/w/index.php?title=Programacion_evolutiva&oldid=22955732>.

[15] Colaboradores de Wikipedia. *Algoritmos genéticos* [en línea]. Wikipedia, La enciclopedia libre, 2012 Disponible en: <http://es.wikipedia.org/w/index.php?title=Algoritmos_geneticos&oldid=61763424>.

[16] Colaboradores de Wikipedia. *Programación genética* [en línea]. Wikipedia, La enciclopedia libre, 2013 en: <http://es.wikipedia.org/w/index.php?title=Programaci%C3%B3n_gen%C3%A9tica&oldid=65623377>.

[17] Colaboradores de Wikipedia. *Distancia de Levenshtein* [en línea]. Wikipedia, La enciclopedia libre, 2013 Disponible en: <http://es.wikipedia.org/w/index.php?title=Distancia_de_Levenshtein&oldid=64701852>.

Este documento esta firmado por

	**Firmante**	CN=tfgm.fi.upm.es, OU=CCFI, O=Facultad de Informatica - UPM, C=ES
	Fecha/Hora	Fri Feb 14 19:47:18 CET 2014
	Emisor del Certificado	EMAILADDRESS=camanager@fi.upm.es, CN=CA Facultad de Informatica, O=Facultad de Informatica - UPM, C=ES
	Numero de Serie	630
	Metodo	urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signature)