Programación Evolutiva
Práctica 3

Roberto Plaza Hermira

Gorka Silva Ramón

Detalles de implementación

1. Gramática Evolutiva

Adaptación del estilo de las gramáticas. Hemos adaptado las gramáticas de ejemplo para poder reconocerlas fácilmente haciendo cambios como:

- En el ejemplo era: (<expr> AND <expr>). Y ahora es: AND (<expr>) (<expr>)
- Ajustes en la disposición de los paréntesis para que todas las gramáticas tengan el mismo formato.

Cruces y mutaciones implementadas.

Cruce OX	Mutación por inserción
Cruce OX-PP	Mutación por intercambio
Cruce CO	Mutación por inversión
	Mutación heurística

Guía de uso.
Para el multiplexor 4 a 1 se pueden utilizar las gramáticas: gramatica1.txt, gramatica3.txt.
Para el multiplexor 8 a 1 es necesario utilizar la gramática: gramática2.txt.

Adaptación para conseguir mejores resultados.
Ya que las ejecuciones con gramática evolutiva no han conseguido tan buenos resultados hemos realizado una pequeña adaptación para que le lleve menos tiempo mejorar los individuos iniciales. Consiste en no permitir que en la población inicial existan individuos de un solo elemento terminal. Ya que estos elementos de un solo terminal al principio pueden conseguir fitness altos y por lo tanto dificultar la evolución.

2. Programación Genética

Implementación del árbol:

El árbol consiste de una clase abstracta “Árbol” completada por herencia con las clases “Nodo” y “Hoja”. La clase “Nodo” agrupa a las expresiones funcionales mientras que la clase “Hoja” agrupa a las expresiones terminales.

Las clases que heredan de “Hoja” son: “HojaA0”, “HojaA1”, “HojaA2”, “HojaD0”, “HojaD1”, “HojaD2”, “HojaD3”, “HojaD4”, “HojaD5”, “HojaD6” y “HojaD7”.

Las clases que heredan de “Nodo” son: “NodoAnd”, “NodoNot”, “NodoIf” y “NodoOr”.

Cada “Árbol” contiene los diferentes datos:

- Lista de hijos
- Padre
- Profundidad: esta variable indica la profundidad máxima de los hijos. Ejemplo: si la profundidad indicada en la interfaz es de 5, la raíz del árbol tendrá profundidad 4, mientras que las hojas tendrán como mínimo 0.

- Niveles hijos: esta variable indica la mayor altura de entre sus hijos. El propio nodo no está incluido. Tiene un valor máximo igual a “Profundidad”.
- Tamaño subárbol: esta variable indica el tamaño del subárbol del que se es la raíz.
- m6: es un boolean que indica si se tiene que ejecutar para un multiplexor de 6 entradas (true) o para uno de 11 entradas (false).

Cruces y mutaciones implementadas.

Cruce por intercambio de subárboles	Mutación por Expansión Mutación por Contracción Mutación Funcional Mutación Hoist Mutación Terminal Mutación por Árbol Mutación por Permutación

Bloating

Se ha implementado la función de bloating mediante el método de “Penalización bien fundamentada”.

Programación Genética

1. Multiplexor 4 a 1. Fitness máximo: 64

Con estas configuraciones la selección y mutación elegidas no afectan significativamente al rendimiento del programa.

Figura 1.
Profundidad = 5
Tipo de generación = Full
Tamaño de población = 100.
Número de generaciones = 100.

Figura 2.
Profundidad = 5
Tipo de generación = Full
Tamaño de población = 100.
Número de generaciones = 100.

En el caso del tipo de generación “Full”, aumentar la profundidad no mejora significativamente el rendimiento, al contrario que en los tipos de generación “Grow”, que obtiene peores resultados con una profundidad de 5, pero al aumentarla a 7-8 mejoran visiblemente. “Ramped and Half” obtiene peores resultados en ambas configuraciones.

Figura 3.
Profundidad = 5
Tipo de generación = Grow
Tamaño de población = 100.
Número de generaciones = 100.

Figura 4.
Profundidad = 7
Tipo de generación = Grow
Tamaño de población = 100.
Número de generaciones = 100.

Si bien con ambas profundidades llegan a una solución óptima, al aumentar esta de 5 a 7, aumenta la frecuencia con la que se llega a la misma.

Figura 5.
Profundidad = 7
Tipo de generación = Ramped and Half
Tamaño de población = 100.
Número de generaciones = 100.

Aumentar el tamaño de la población y el número de generaciones a 200 mejora en algunos casos la solución , haciendo que lleguen al óptimo de 64, pero en la mayoría de las ocasiones no tiene efecto, ya que generalmente converge antes de la generación 80.

2. Multiplexor 8 a 1. Fitness máximo: 2048

No hemos podido hacer todas las pruebas que nos gustaría ya que cada ejecución tarda alrededor de diez minutos para 200 individuos y 200 generaciones. Como se puede observar en las imágenes, el programa no converge, por lo que aumentar el número de generaciones mejoraría significativamente la solución.

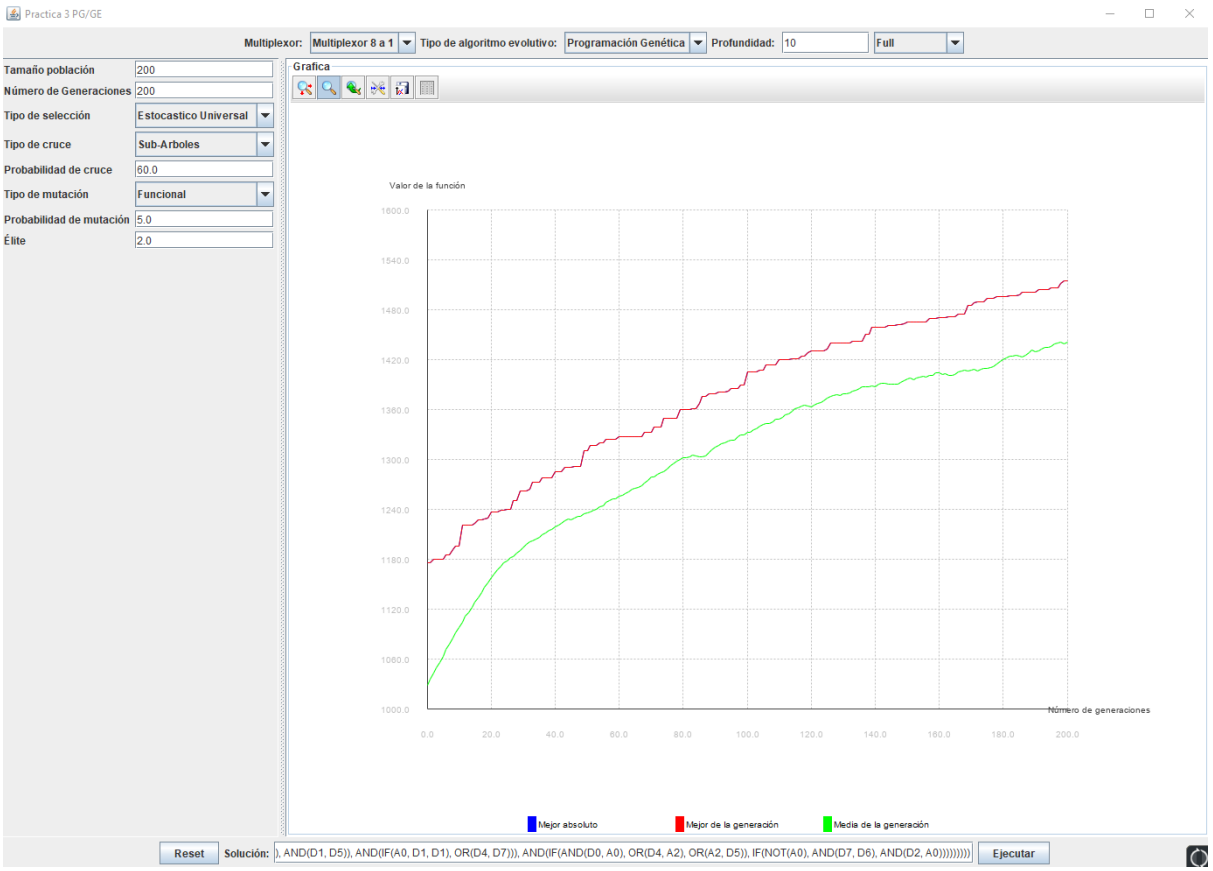

Figura 6.
Profundidad = 10
Tipo de generación = Full
Tamaño de población = 200.
Número de generaciones = 200.

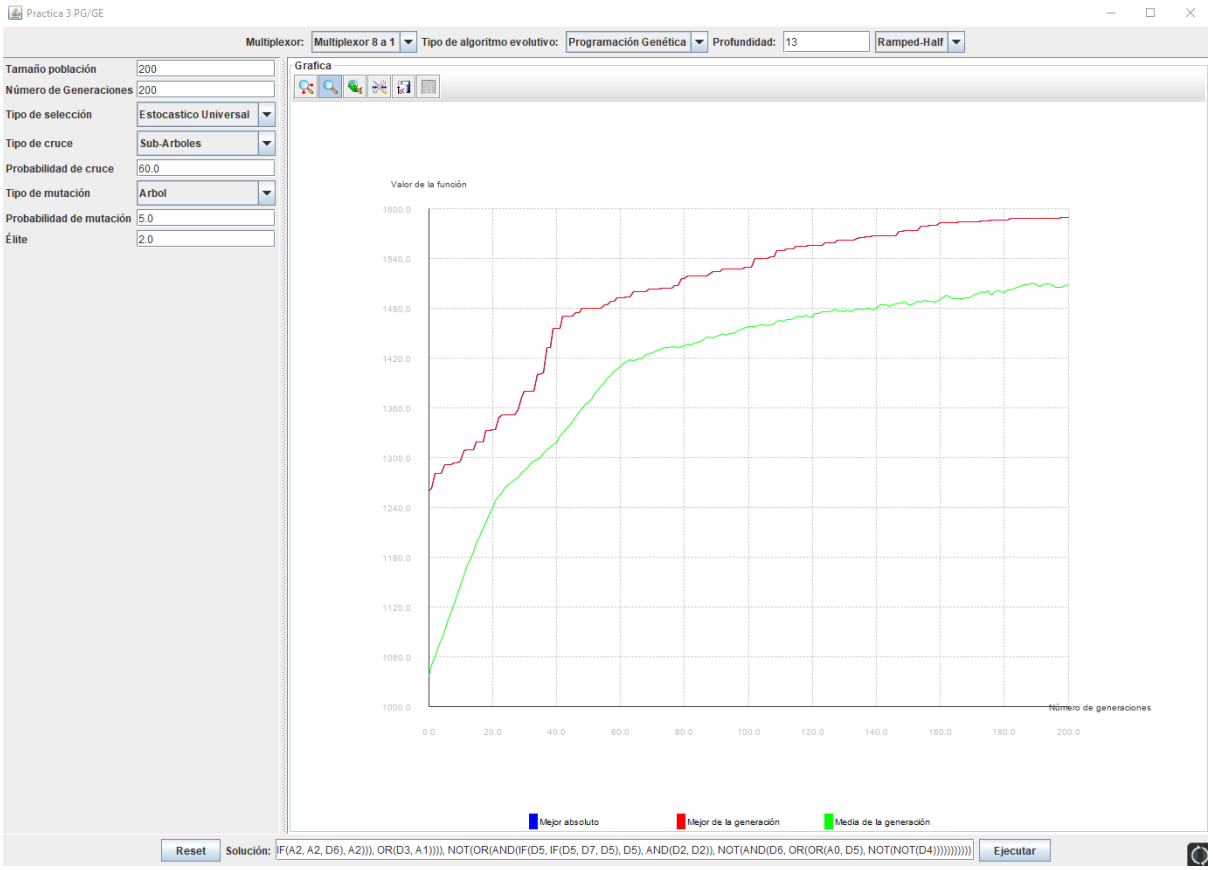

Figura 7.
Profundidad = 13
Tipo de generación = Ramped and Half
Tamaño de población = 200.
Número de generaciones = 200.

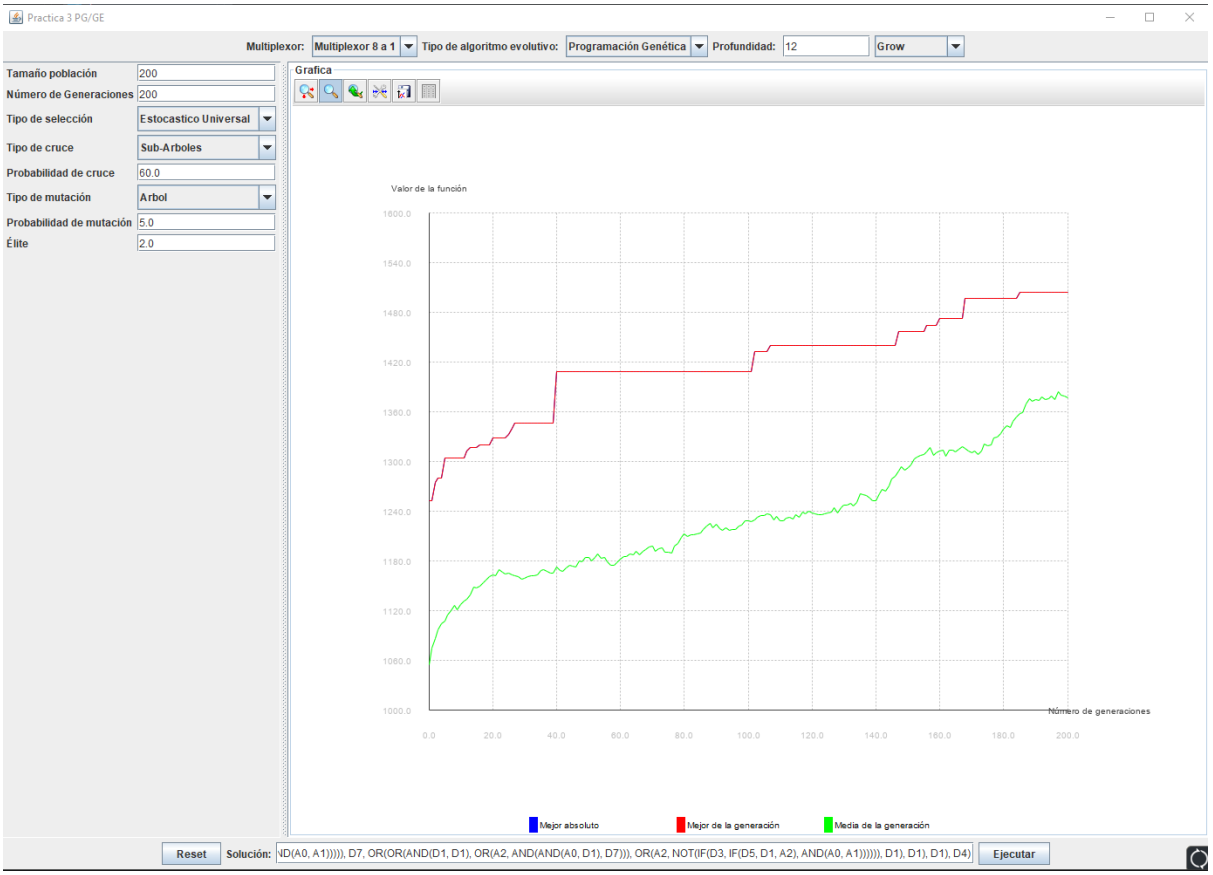

Figura 8.
Profundidad = 12
Tipo de generación = Grow
Tamaño de población = 200.
Número de generaciones = 200.

Gramática Evolutiva

1. Multiplexor 4 a 1. Fitness máximo: 64

El cruce Monopunto tiene el inconveniente de que, para individuos con longitud de cromosoma larga y longitud de solución corta, puede darse el caso de que el cruce no cambie nada. Esto solo puede ocurrir si los dos individuos cumplen lo anterior y en el punto de corte está situado en una posición del cromosoma que es mayor que la solución que representan. Esto se produce debido a que el cruce se hace intercambiando la primera parte del cromosoma. También se podría haber utilizado el intercambio en la segunda parte, pero hemos comprobado que este problema (que el cruce no cambia nada) se produce con mayor frecuencia. Debido a todo lo anterior, los individuos con longitud muy larga tienen esta restricción que hace que se produzcan menos cruces, lo que a su vez hace que los individuos converjan de forma más rápida sobre una solución local.

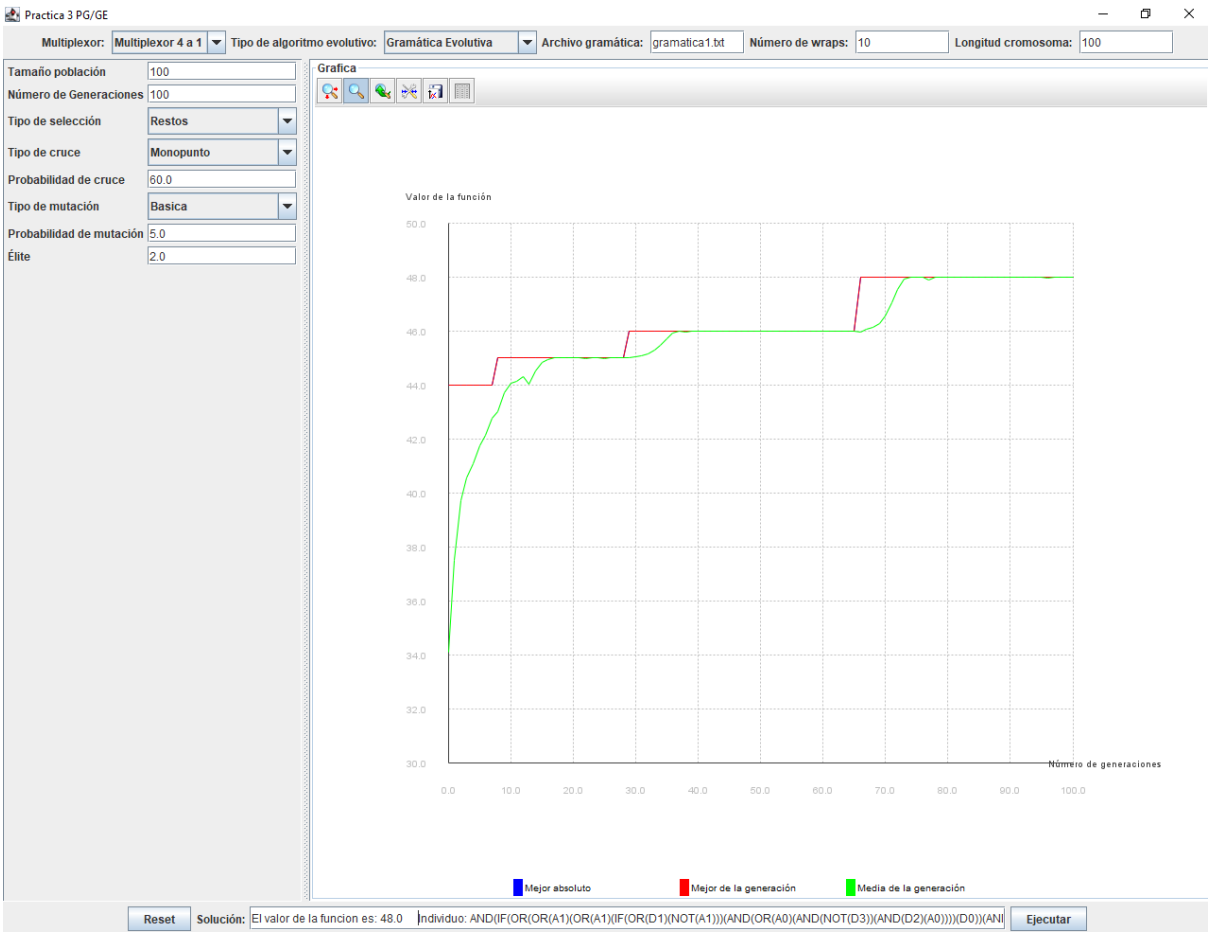

Figura 9.
Número de wraps = 10.
Longitud del cromosoma = 100.
Tamaño de población = 100.
Número de generaciones = 100.

La Figura 9, además de mostrar el problema de la convergencia del cruce Monopunto, muestra algo que ocurre con bastante frecuencia en este problema: cuando se encuentra un

individuo con un valor alto de fitness al principio, se converge rápidamente sobre él. Cuando ocurre esto es muy difícil que se encuentre la solución del problema.

Este problema no encuentra resultados significativos con tamaño de población = 100 y número de generaciones = 100. Cuando utilizamos tamaño de población = 100 y número de generaciones = 300 empezamos a encontrar resultados mejores, llegando a la solución.

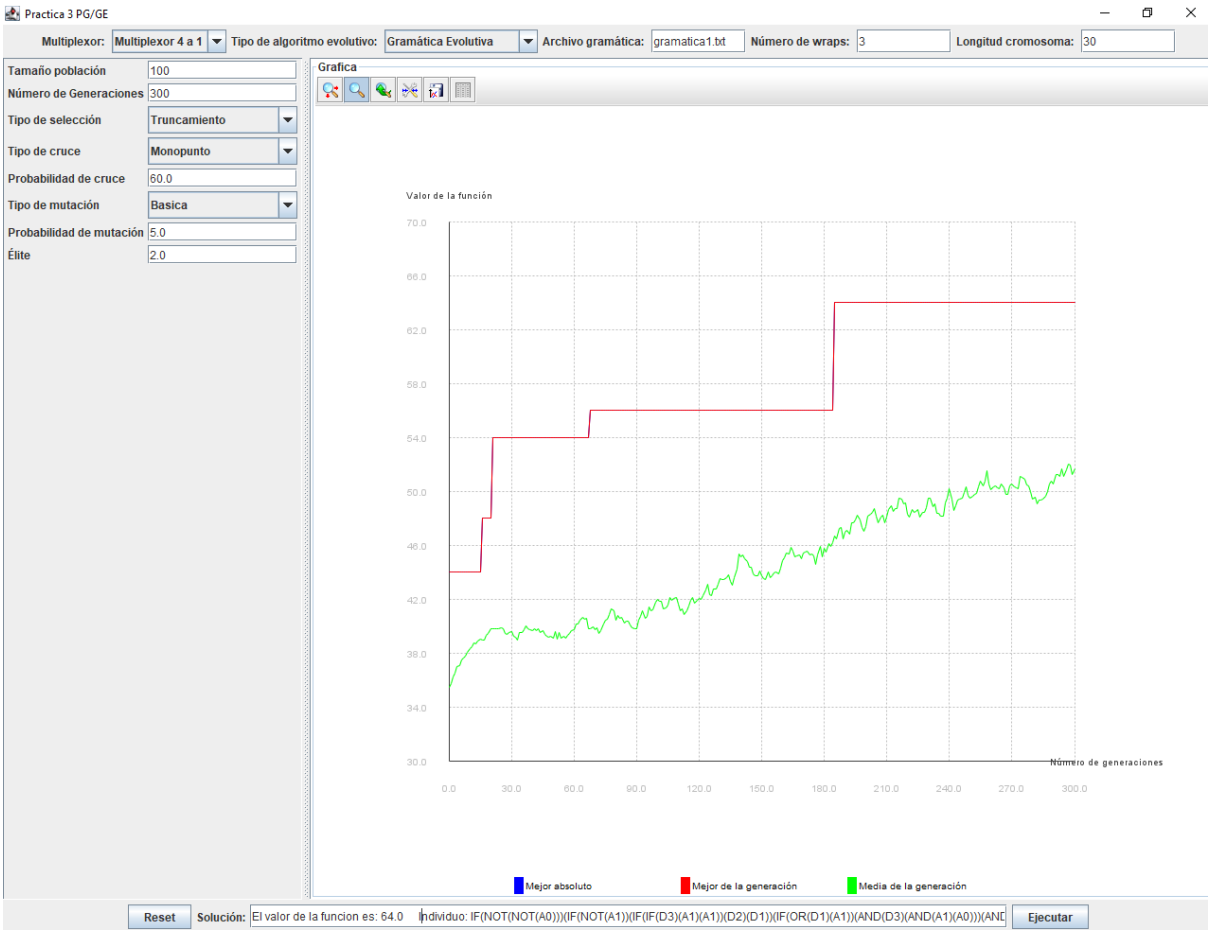

Figura 10.
Número de wraps = 3.
Longitud del cromosoma = 30.
Tamaño de población = 100.
Número de generaciones = 300.

Al utilizar diferentes selecciones se observan resultados muy dispares. Para simplificar las explicaciones he dividido a las selecciones en dos grupos.

Grupo 1	Grupo 2
Ruleta	Restos

Truncamiento	Torneo determinístico
Estocástico universal	Torneo probabilístico
	Ranking

En general las selecciones del grupo 1 han conseguido mejores resultados. El problema de las selecciones del grupo 2 es que convergen muy rápido sobre una solución local. En la Figura 12 se puede ver un ejemplo de la selección Torneo probabilístico. Y en la Figura 13 un ejemplo con la selección Truncamiento, que es la selección que ha conseguido mejores resultados.

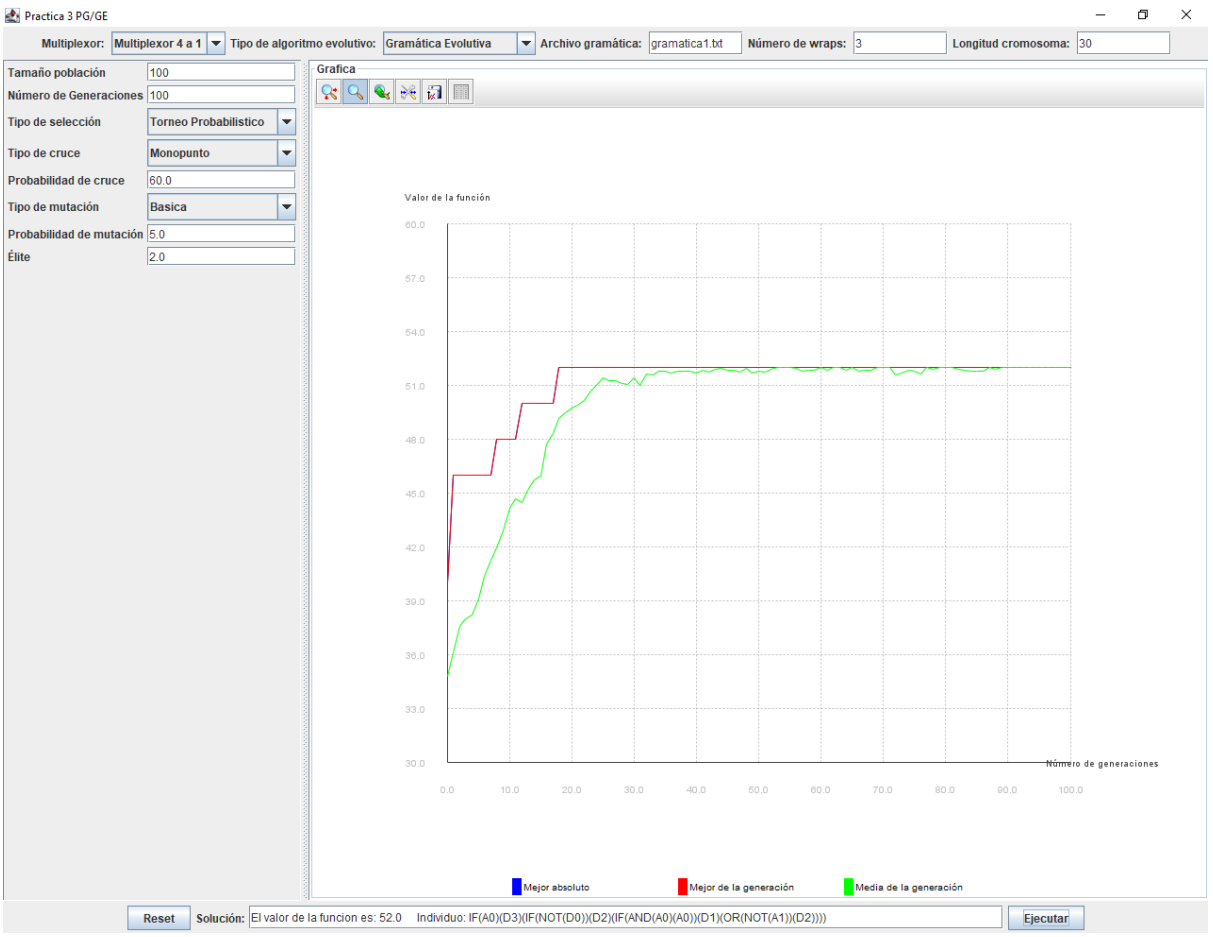

Figura 12.
Número de wraps = 3.
Longitud del cromosoma = 30.
Tamaño de población = 100.
Número de generaciones = 100.

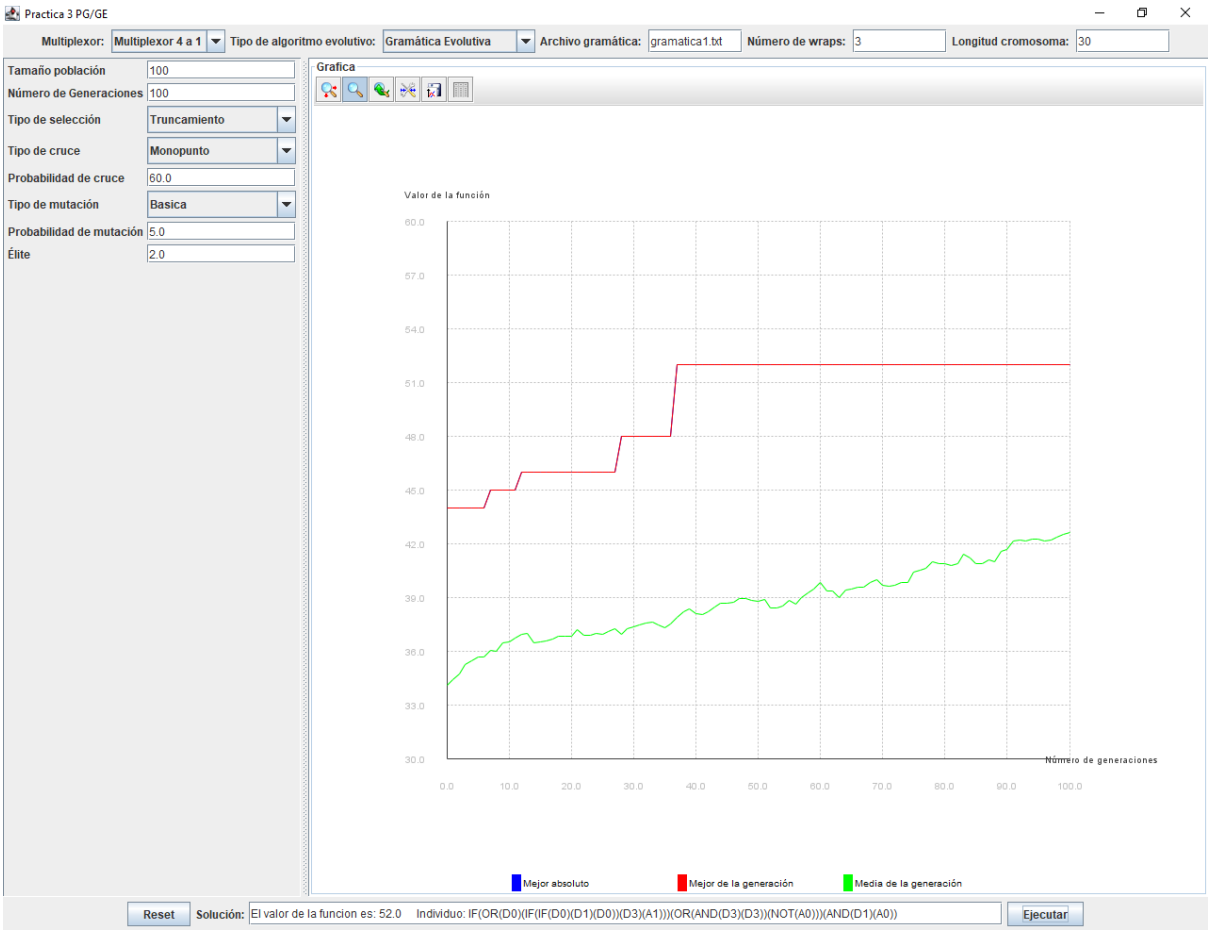

Figura 13.
Número de wraps = 3.
Longitud del cromosoma = 30.
Tamaño de población = 100.
Número de generaciones = 100.

2. Multiplexor 8 a 1. Fitness máximo: 2048

Hemos decidido realizar ejecuciones con la selección que ha dado mejores resultados: Truncamiento. Los resultados con tamaño de población 100 y número de generaciones 100 no son reveladores, pero sí que se observa una progresión.

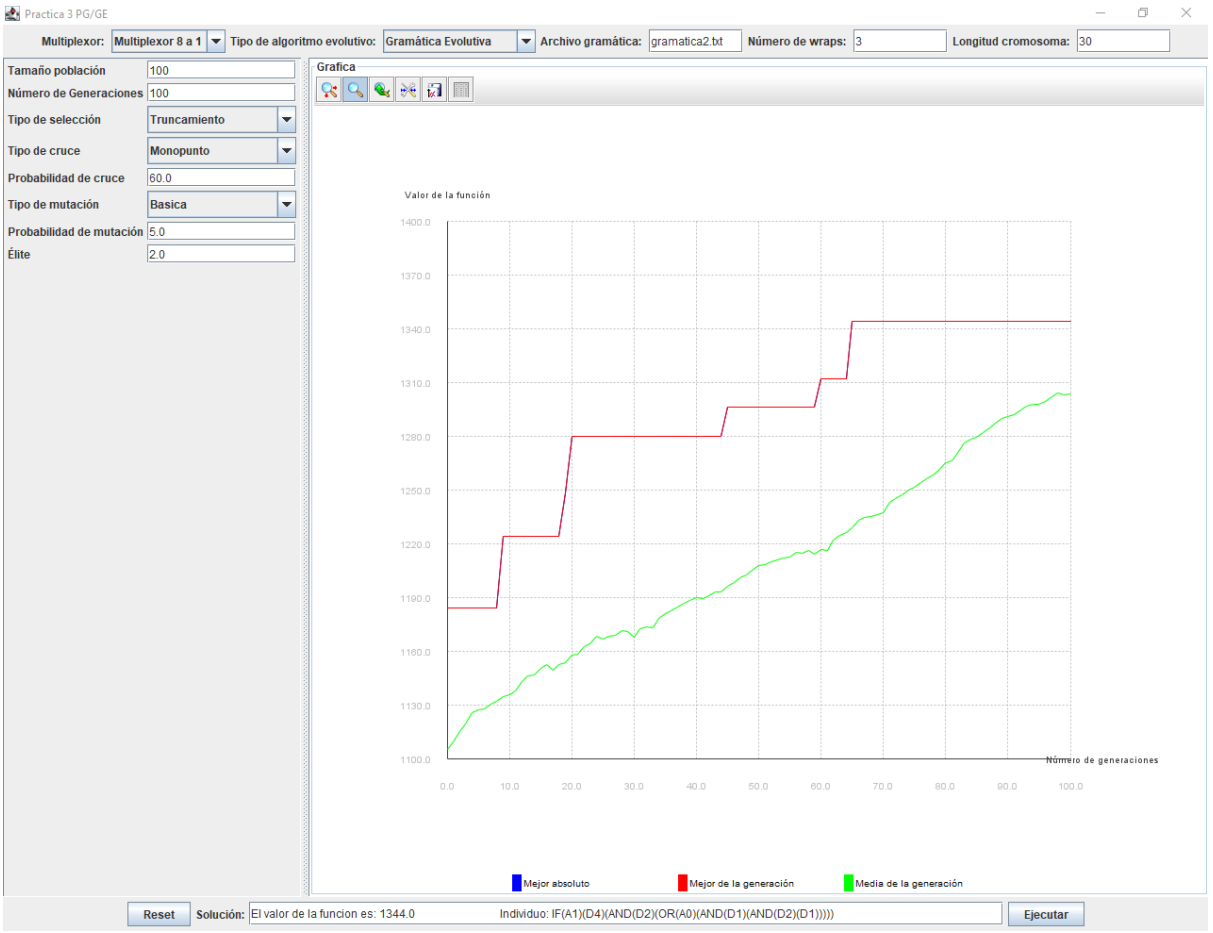

Figura 14.
Número de wraps = 3.
Longitud del cromosoma = 30.
Tamaño de población = 100.
Número de generaciones = 100.

En este problema se sigue el mismo comportamiento que en la Programación Genética. La media se mantiene bastante cerca de las líneas con los mejores globales y se observa mejor la progresión que en el problema del multiplexor 4 a 1. Con 300 generaciones se obtienen mejores resultados, pero todavía no está cerca de encontrar la solución, como se muestra en la Figura 15.

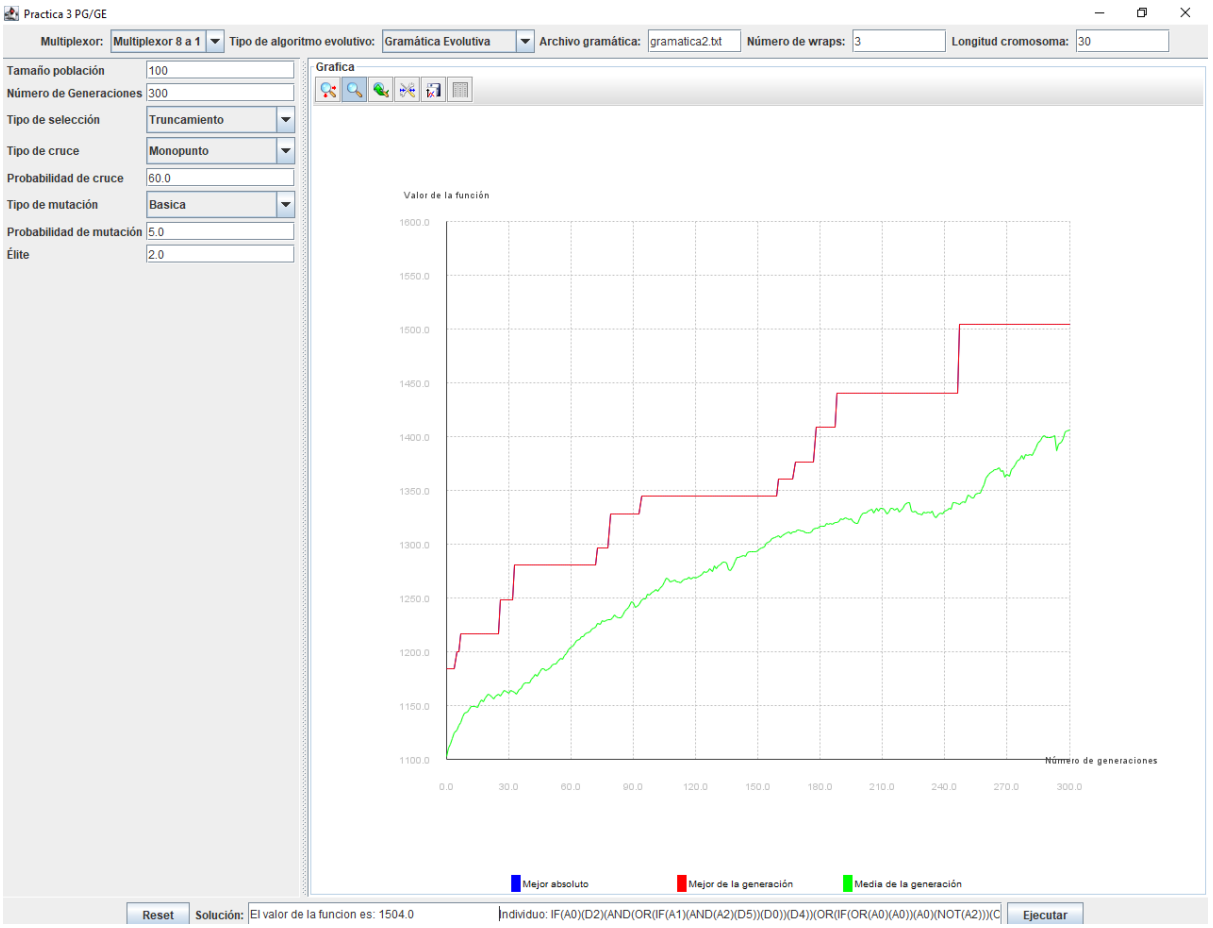

Figura 15.
Número de wraps = 3.
Longitud del cromosoma = 30.
Tamaño de población = 100.
Número de generaciones = 300.

Al igual que ocurría en el multiplexor 4 a 1, las selecciones del grupo 2 siguen provocando una convergencia de la media de las generaciones, como se muestra en la Figura 16. En este problema se obtienen peores resultados con este tipo de selecciones. En Programación Genética podíamos observar que, aunque la media de generaciones converja rápidamente, se podía seguir produciendo un progreso. En este caso, la convergencia de generaciones produce un estancamiento.

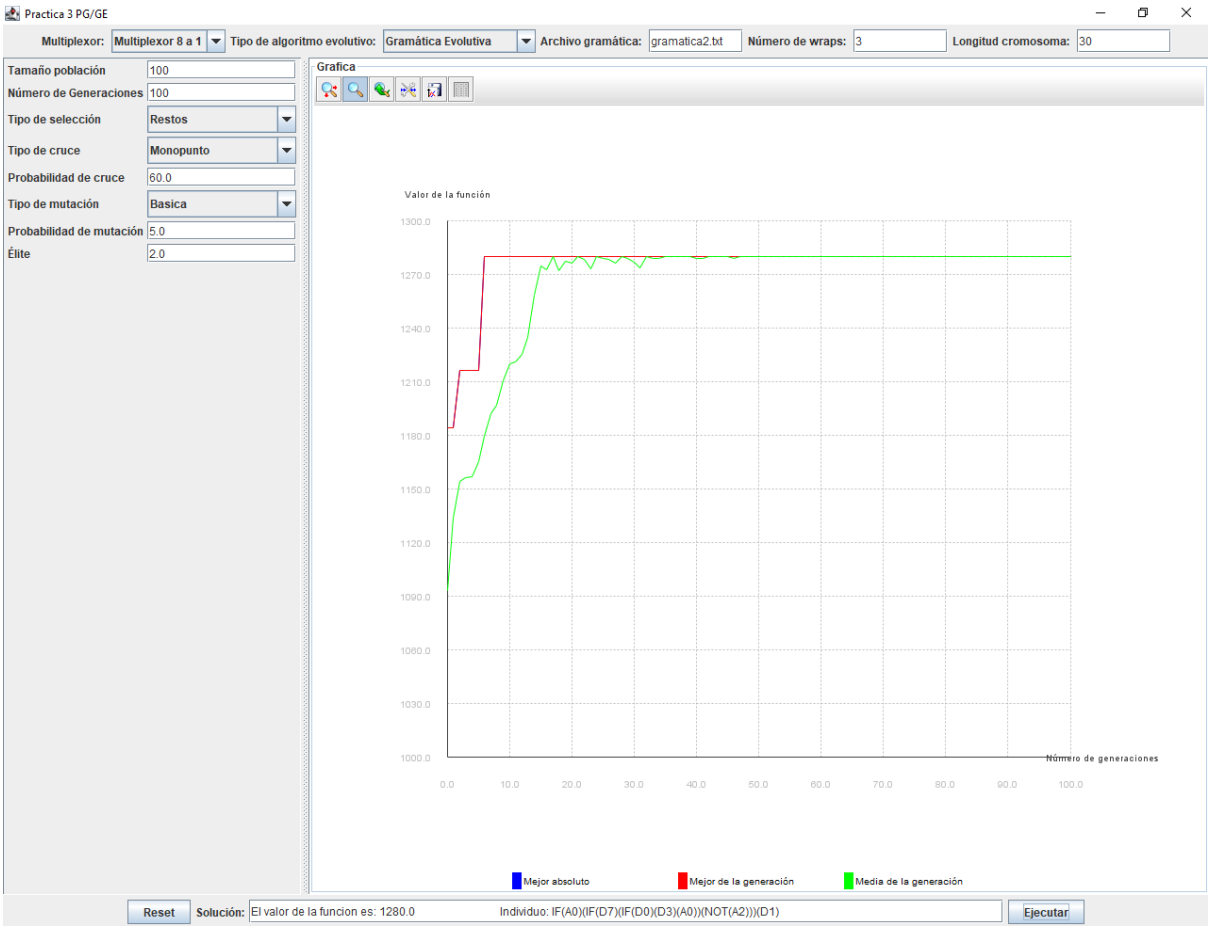

Figura 16.
Número de wraps = 3.
Longitud del cromosoma = 30.
Tamaño de población = 100.
Número de generaciones = 100.

Reparto de tareas

Roberto se ha encargado principalmente de:

- Programación Genética

Gorka se ha encargado principalmente de:

- Gramática Evolutiva
- Adaptación Interfaz GUI

La memoria se ha hecho en cooperación.