MEMORIA PRÁCTICA 3 PEV

Grupo 6 Pablo Fernández Jara y Víctor Gómez-Jareño Guerrero

Selección:

No hemos cambiado nada de las anteriores prácticas, los métodos de selección son las siguientes:

● **Estocástica**: Generamos un número a partir de 1/tamaño de la población. Calculamos una distancia entre las marcas son 1/N, siendo N el tamaño de la población. Cogemos los individuos sumando la distancia al número aleatorio hasta que alcanzamos un número determinado de individuos escogidos.

● **Ranking**: Hacemos una clasificación acorde al fitness decreciente de los individuos y calculamos una probabilidad de selección. Si lo cumple, se selecciona.

● **Restos**: De un individuo escogemos puntuación*0,25 copias. Los que faltan se escogen por otro método de selección. En nuestro caso escogemos el método de Ruleta.

● **Ruleta**: Parecido al estocástica, pero en este escogemos el tamaño de cada individuo en la franja depende de su puntuación acumulada. Después se van a escogiendo al azar lugares en el segmento recién dividido y se escogen.

● **Torneo Determinístico**: Se escogen dos individuos al azar y seleccionamos el mejor de esos dos. Y así hasta rellenar el tamaño de la población.

● **Torneo Probabilístico:** Es igual que el anterior pero en vez de seleccionar el mejor de los dos, creamos un número aleatorio y si es mayor que 0.75 (número establecido por nosotros), escogemos el mejor de los dos, si no, el peor.

● **Truncamiento**: Ordenamos los individuos por su fitness, cogemos el mismo individuo un número 1/0.5 de veces. Así hasta rellenar toda la población de nuevo. De esta manera escogemos solo los mejores.

Inicialización:

Hemos implementado los tres tipos de inicializaciones vistos en teoría, antes de todo, comprobamos si usamos la operación **IF** o no en esta inicialización.

La **inicialización completa** que baja desde la raíz de árbol hasta completar todos los nodos que son funciones, una vez completados se ponen los nodos terminales (datos).

La **inicialización creciente** cogemos los nodos de funciones y terminales y completamos el árbol hasta una profundidad dada. A partir de este punto funciona igual que el anterior, quedando de esta forma un árbol más regular.

Por último hemos implementado la inicialización **Ramped and half** que es una mezcla entre las dos. Divide el tamaño de la población en grupos y cada grupo utiliza valores de profundidad diferentes y la mitad utiliza la inicialización creciente y la otra mitad la completa. El resultado de esta inicialización es muy eficaz porque creas árboles muy diversos y de distintos tamaños. Siendo los más regulares los creados por la inicialización creciente.

Mutación:

Hemos visto en teoría distintos tipos de mutaciones, debido a la diversidad de estas hemos decidido implementar el máximo posible. Todas estas dependen de una probabilidad de mutación, si no la cumplen, no se ejecutan. Si mutan, hay que evaluar el árbol de nuevo porque habrá cambiado su fitness.

● **Hoist**: Elegimos una posición al azar del árbol y eliminamos todo el árbol restante exceptuando el subárbol que nace a raíz de la posición que hemos elegido. Solo puede ser una función lo que hayamos escogido aleatoriamente, para no provocar un subárbol con un único elemento terminal.

● **Expansión**: Como bien indica su nombre consiste en expandir el árbol a partir de un nodo terminal. Creamos un subárbol nuevo con inicialización creciente y lo implantamos en esa posición.

Después de esto hay que recalcular el número de nodos del árbol debido a que habrá aumentado.

● **Contracción**: Al revés que la mutación de expansión, escogemos un nodo al azar y eliminamos el subárbol que contenga. Para ello, creamos un árbol de un único elemento solamente y lo implantamos en dicha posición elegida. Al igual que en la anterior, tenemos que recalcular el número de nodos.

● **Terminal simple**: Seleccionamos al azar un nodo terminal del árbol y lo sustituimos por otro valor aleatorio que sea posible.

● **Funcional simple**: muy parecida a la terminal simple, pero seleccionando un nodo que contenga una función ahora e intercambiandolo por otro tipo de función aleatoria dentro de las posibles.

● **De árbol**: Seleccionamos un subárbol y lo eliminamos para meter un subárbol totalmente nuevo.

● **Permutación**: Intercambiamos dos subárboles para que su operación la haga a la inversa.

Bloating:

El **Bloating** sirve sobre todo para evitar que, a medida que las generaciones avancen, el tamaño de los individuos crezca. Existen dos métodos que hemos visto en teoría y hemos implementado estos mismos. Para poder llevarlos a cabo, hemos visto que teníamos que calcular la media de nodos que existían en total, y la media del fitness_bruto de cada población.

● Método **Tarpeian**: Para este método necesitamos solamente la media de nodos que hemos calculado previamente y un 2 que le pasamos para calcular la probabilidad de que ese individuo muera (1/n). Comprobamos que la longitud sea mayor que la media y entonces generamos un número aleatorio y si cumple esta probabilidad, generamos el individuo de nuevo. Puede ser que sobreviva y entonces se reproduce y el tamaño medio de la población se verá incrementado.

● Método de **penalización**: Sacamos la variancia y la covarianza con estas fórmulas: cov += (c.getFitness_bruto() - media) * (c.getArbol().getNumNodos() - mediaTam); var += (c.getArbol().getNumNodos() - mediaTam) * (c.getArbol().getNumNodos() - mediaTam); Hacemos esto para todo el tamaño de población y después lo dividimos entre el Tam Población para sacar la media. Dividimos la cov entre la var y así obtenemos la k para nuestra fórmula: double fitness = c.getFitness_bruto() - k * c.getArbol().getNumNodos(); Gracias a esto conseguimos el bloating necesario para que el tamaño medio de los individuos sea más o menos el mismo y esté equilibrado.

Cruce:

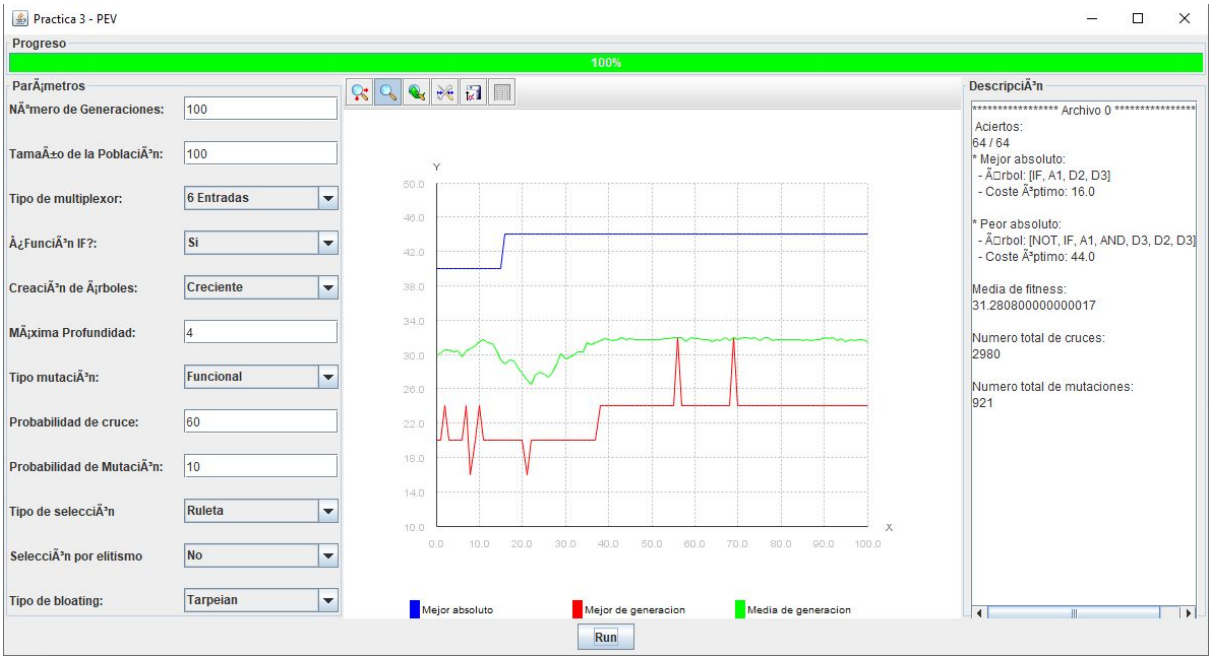
En esta práctica sólo existe un tipo de **cruce**. Hemos implementado esto de la siguiente manera. Seleccionamos los nodos más relevantes según la probabilidad 0.9 y se cruzaran en una función, el resto se cruzará en un terminal. En caso de no cumplir la probabilidad, se cruzará por terminal. Después obtenemos los puntos de cruce a través de los nodos seleccionados. Copiamos los hijos en los padres y cortamos por donde haya salido. Calculamos de nuevo los nodos y evaluamos.

Árboles:

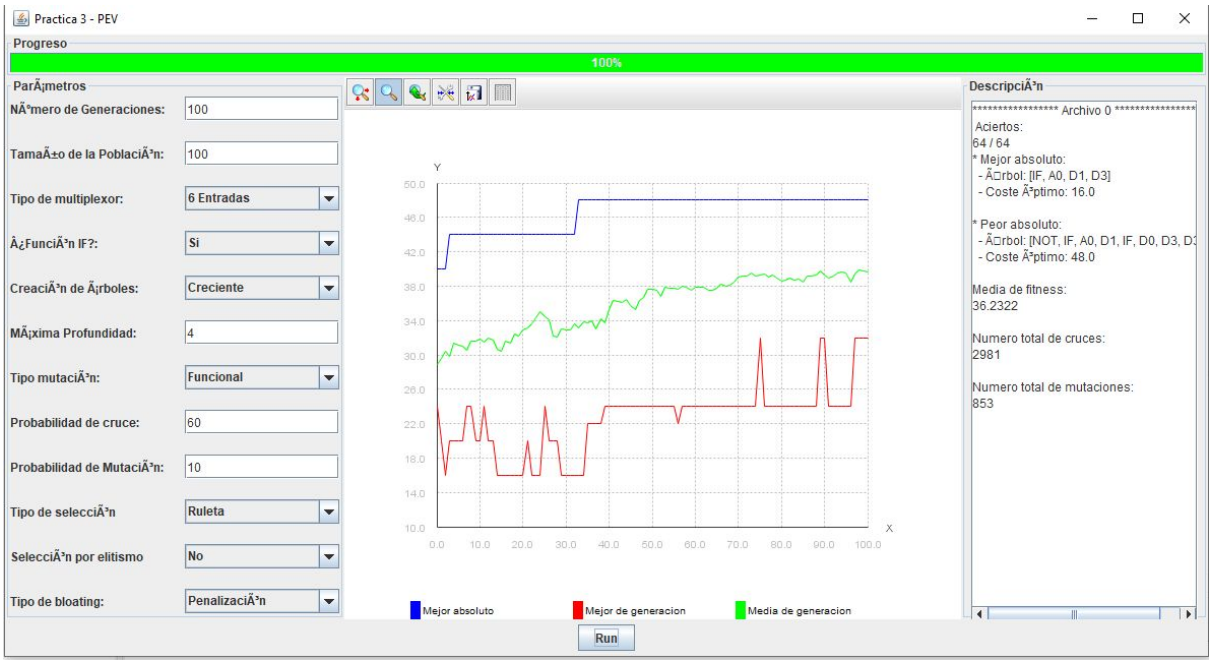
Hemos cogido los 5 mejores de nuestras pruebas. Hemos analizado esta práctica como una función de maximización, por tanto contamos el número de aciertos, intentado que sea el máximo posible. Lógicamente es 64 o 2048, dependiendo del multiplexor.

Multiplexor de 6:

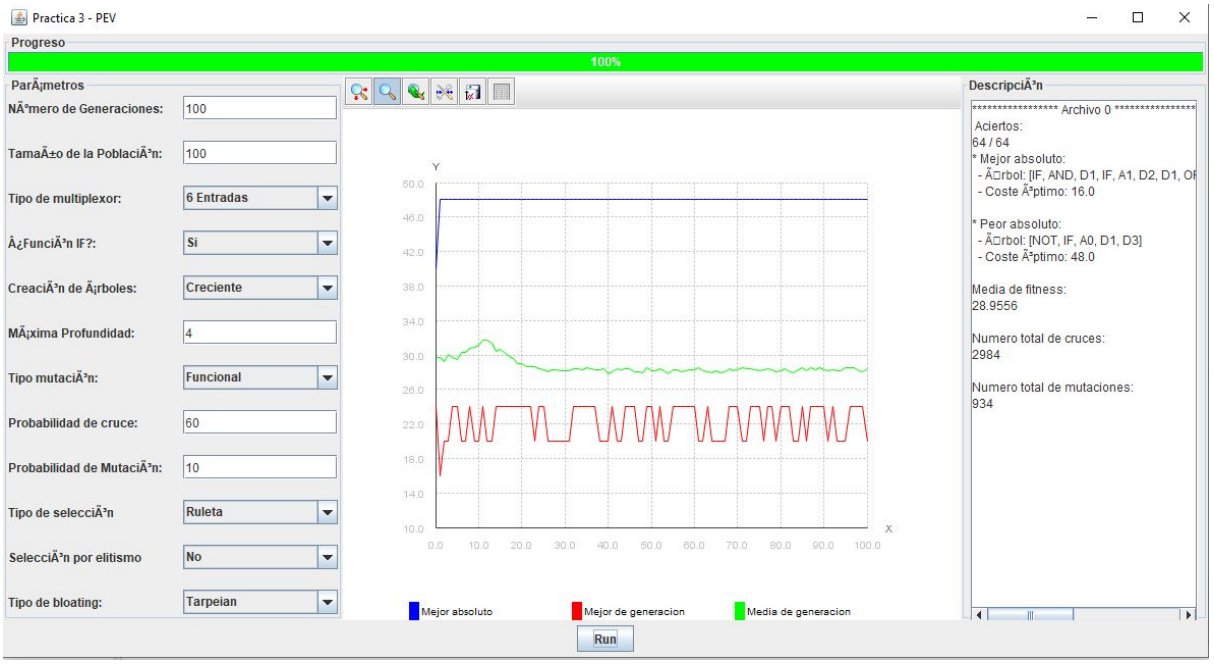
Arbol con 64 aciertos: [IF, A1, D2, D3]
Media de fitness: 31,28
Mutaciones: 921
Cruces: 2980
Hemos observado que la mutación funcional con If da valores menores. La siguiente prueba es la misma pero variando la mutación y se observa que el número de aciertos es menor.



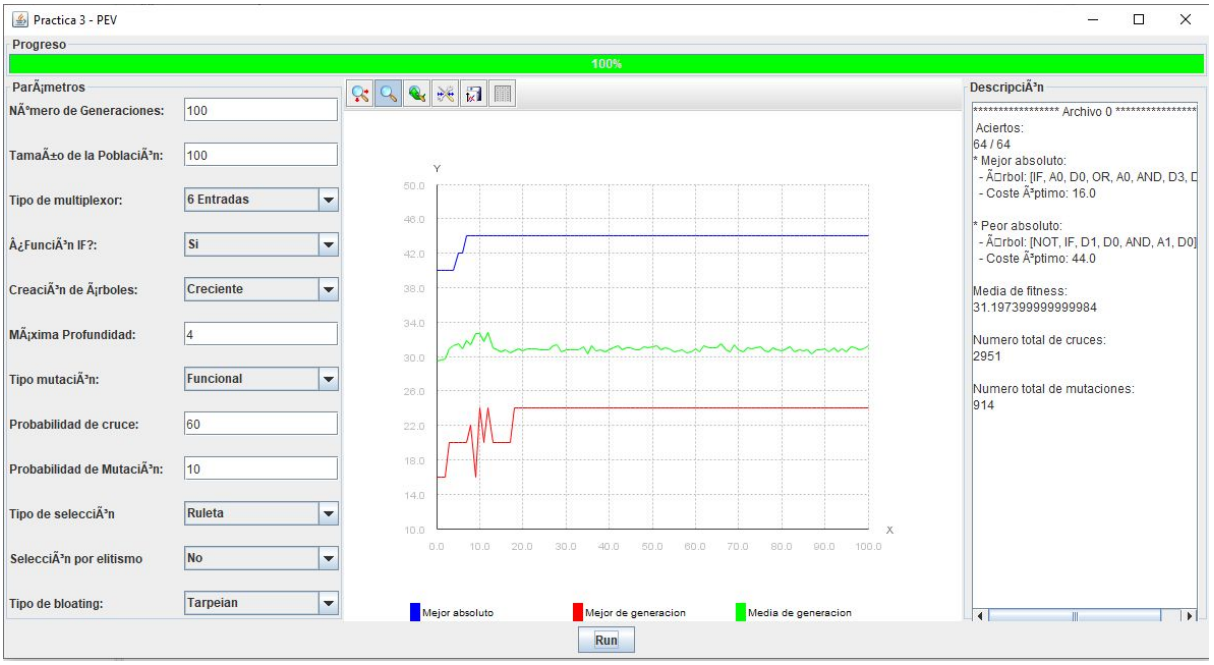
Arbol con 64 aciertos: [IF, A0, D1, D3]
Media de fitness: 36.23
Mutaciones: 853
Cruces: 2981
Comparando con la anterior prueba, vemos que solo varía el método de mutación, por tanto observamos que el Funcional simple es más eficaz.



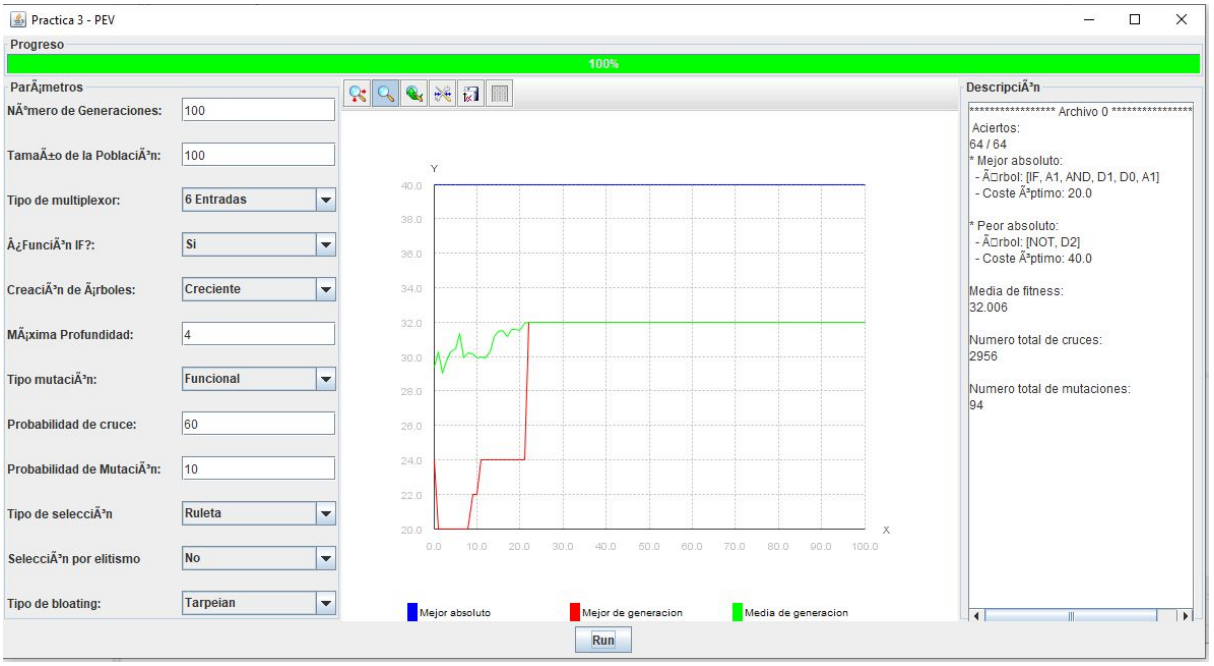
Arbol con 64 aciertos: [IF,AND, D1, IF, A1, D2, D1, OR, D2, D1]
Media de fitness: 28,95
Mutaciones: 934
Cruces: 2984



Arbol con 64 aciertos: [IF, A0, D0, OR, A0, AND, D3, D1]
Media de fitness: 21,19
Mutaciones: 914
Cruces: 2951

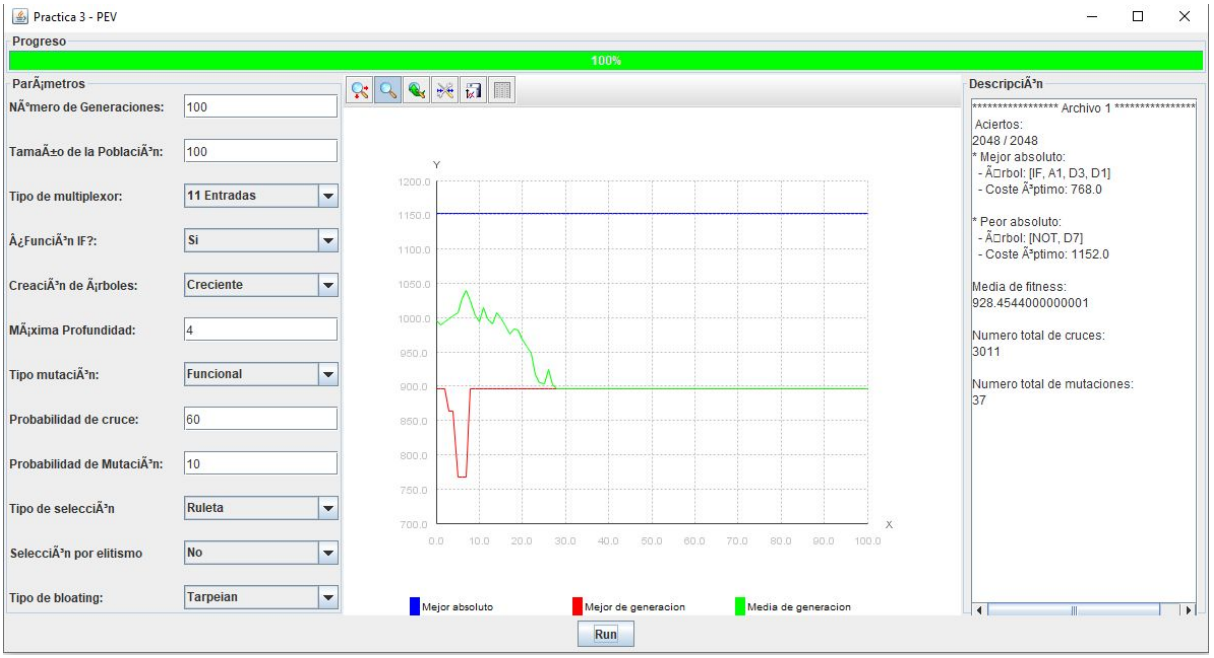


Arbol con 64 aciertos: [IF, A1, AND, D1, D0. A1]
Media de fitness: 32, 006
Mutaciones: 94
Cruces: 2956

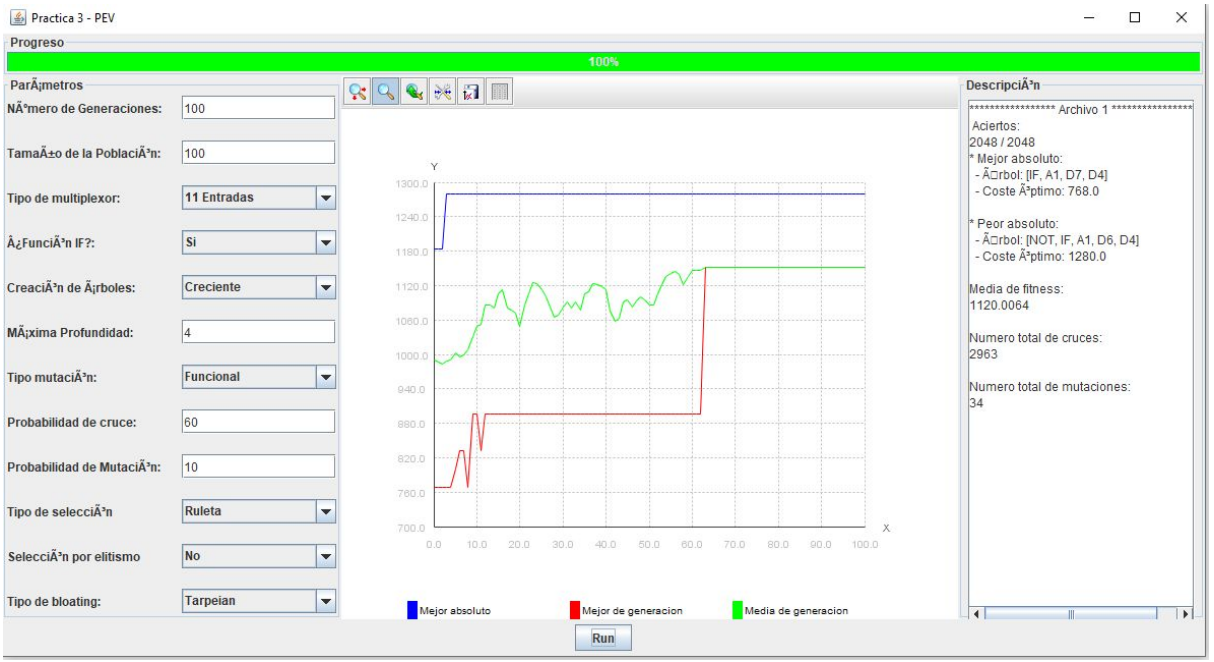


Multiplexor de 11:

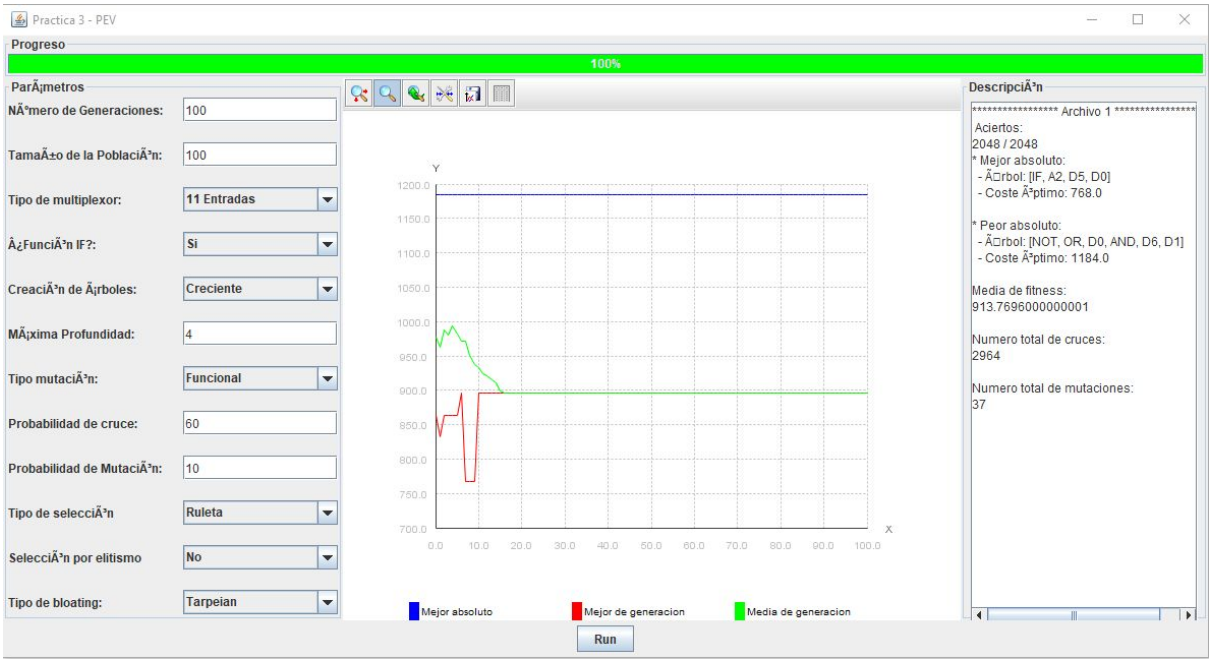
Arbol con 2048 aciertos: [IF, A1, D3, D1]
Media de fitness: 928,45
Mutaciones: 37
Cruces: 3011



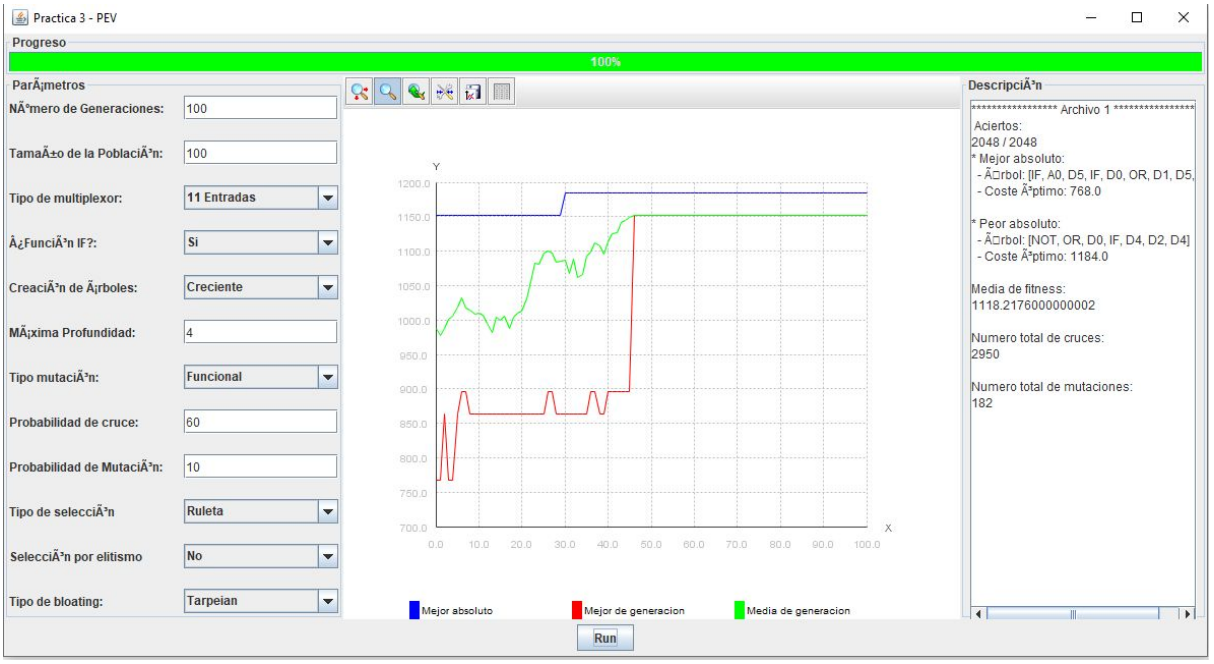
Arbol con 2048 aciertos: [IF,A1, D7, D4]
Media de fitness: 1120,0064
Mutaciones: 34
Cruces: 2963



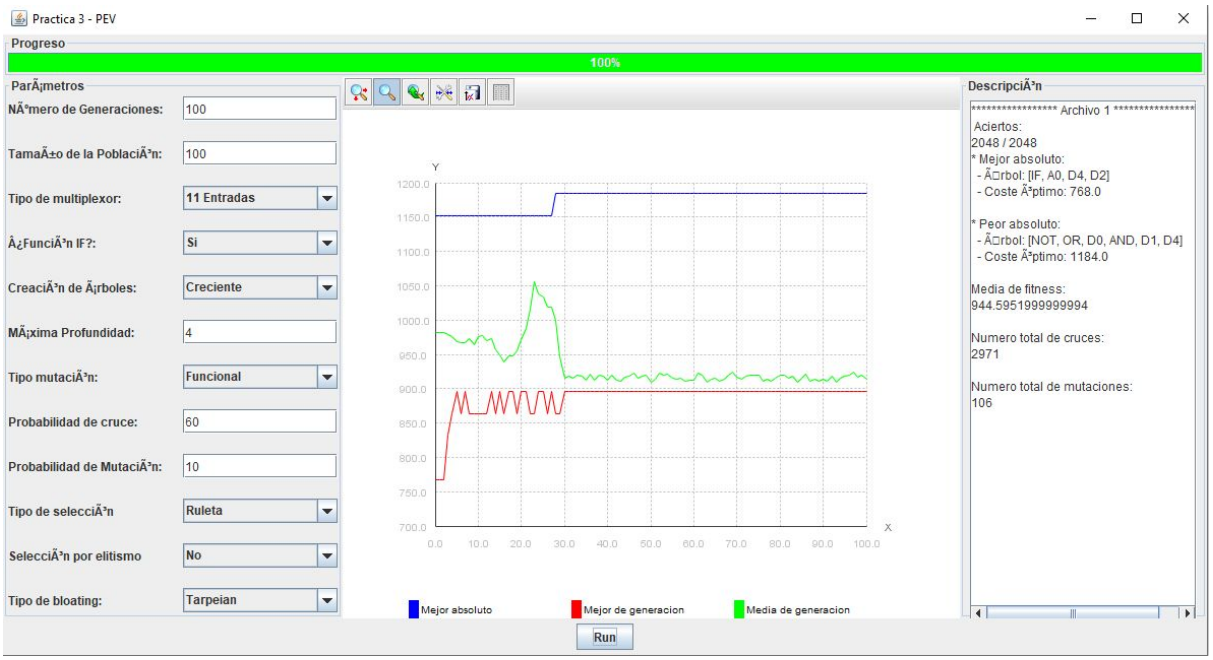
Arbol con 2048 aciertos: [IF, A2, D5, D0]]
Media de fitness: 913,76
Mutaciones: 37
Cruces: 2964



Arbol con 2048 aciertos: [IF, A0, D5, IF, D0, OR, D1, D5, AND, D1, D7]
Media de fitness: 1118,217
Mutaciones: 182
Cruces: 2950



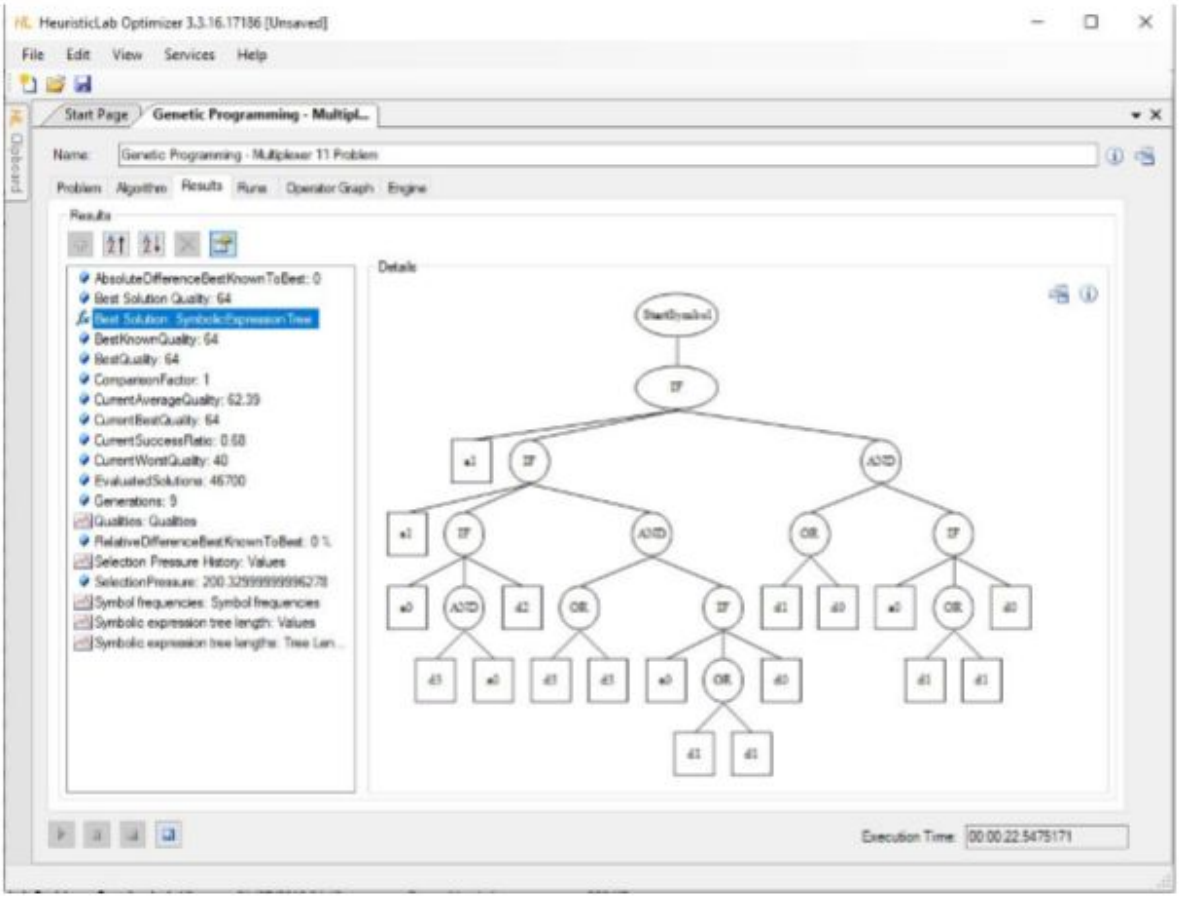
Arbol con 2048 aciertos: [IF, A0,D4, D2]
Media de fitness: 944,595
Mutaciones: 106
Cruces: 2971


Practica 3 - PEV
Progreso
100%
ParÃ¡metros
NÃºmero de Generaciones:
100
TamaÃ±o de la PoblaciÃ³n:
100
Tipo de multiplexor:
11 Entradas
Â¿FunciÃ³n IF?:
Si
CreaciÃ³n de Ã¡rboles:
Creciente
MÃ¡xima Profundidad:
4
Tipo mutaciÃ³n:
Funcional
Probabilidad de cruce:
60
Probabilidad de MutaciÃ³n:
10
Tipo de selecciÃ³n
Ruleta
SelecciÃ³n por elitismo
No
Tipo de bloating:
Tarpeian
Y
X
Mejor absoluto
Mejor de generacion
Media de generacion
Run
DescripciÃ³n
********** Archivo 1 **********
Aciertos:
2048 / 2048
* Mejor absoluto:
- Ã□rbol: [IF, A0, D4, D2]
- Coste Ã³ptimo: 768.0
* Peor absoluto:
- Ã□rbol: [NOT, OR, D0, AND, D1, D4]
- Coste Ã³ptimo: 1184.0
Media de fitness:
944.5951999999994
Numero total de cruces:
2971
Numero total de mutaciones:
106

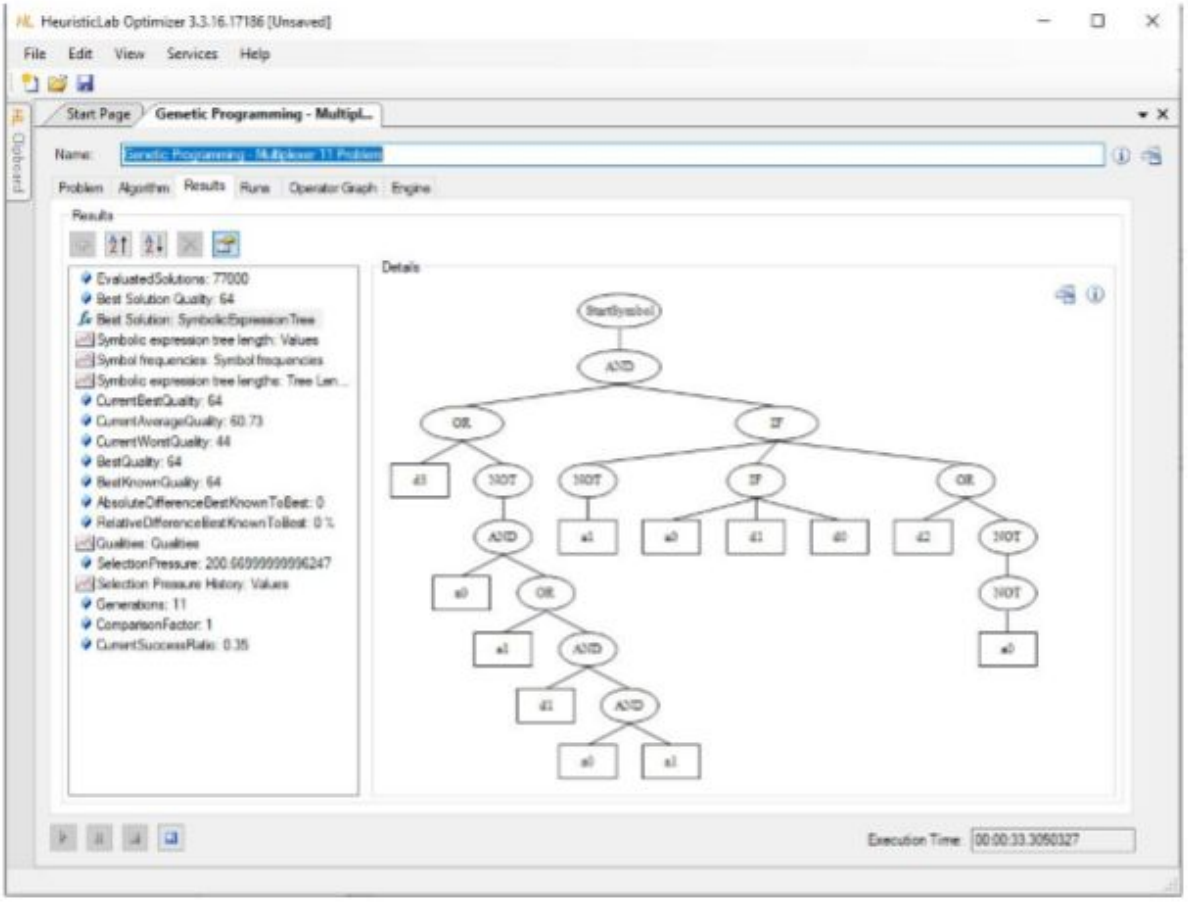
Heuristic Lab:

Multiplexor de 6 entradas:

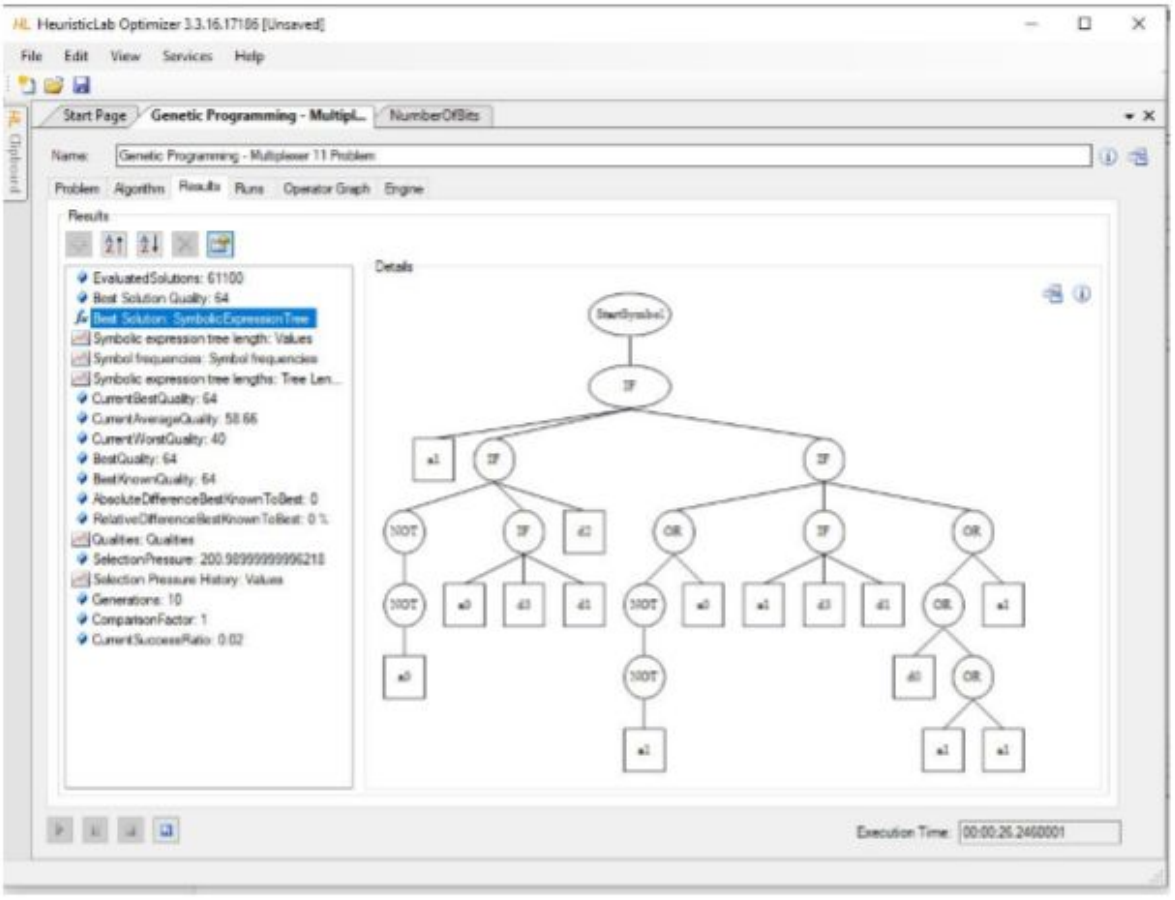
El número máximo de aciertos es 64. En este primer ejemplo, con los parámetros por defecto, vemos que sale el número máximo de aciertos.



Probando otra vez vemos que llega de nuevo al número máximo de aciertos.

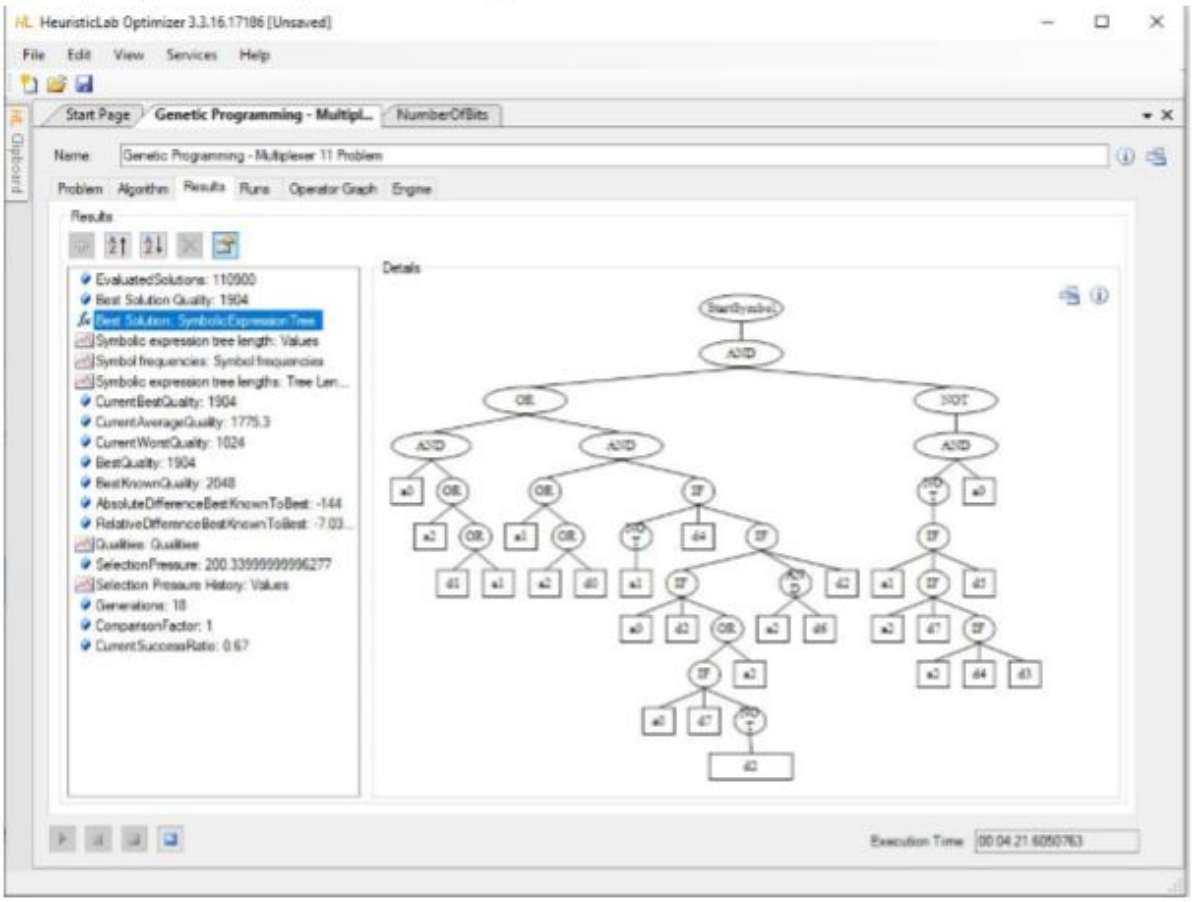


Y tras muchos ejemplos hemos concluido que siempre, o casi siempre, llega al número máximo de aciertos. Aunque salen árboles algo grandes, comparados a los nuestros de la práctica.

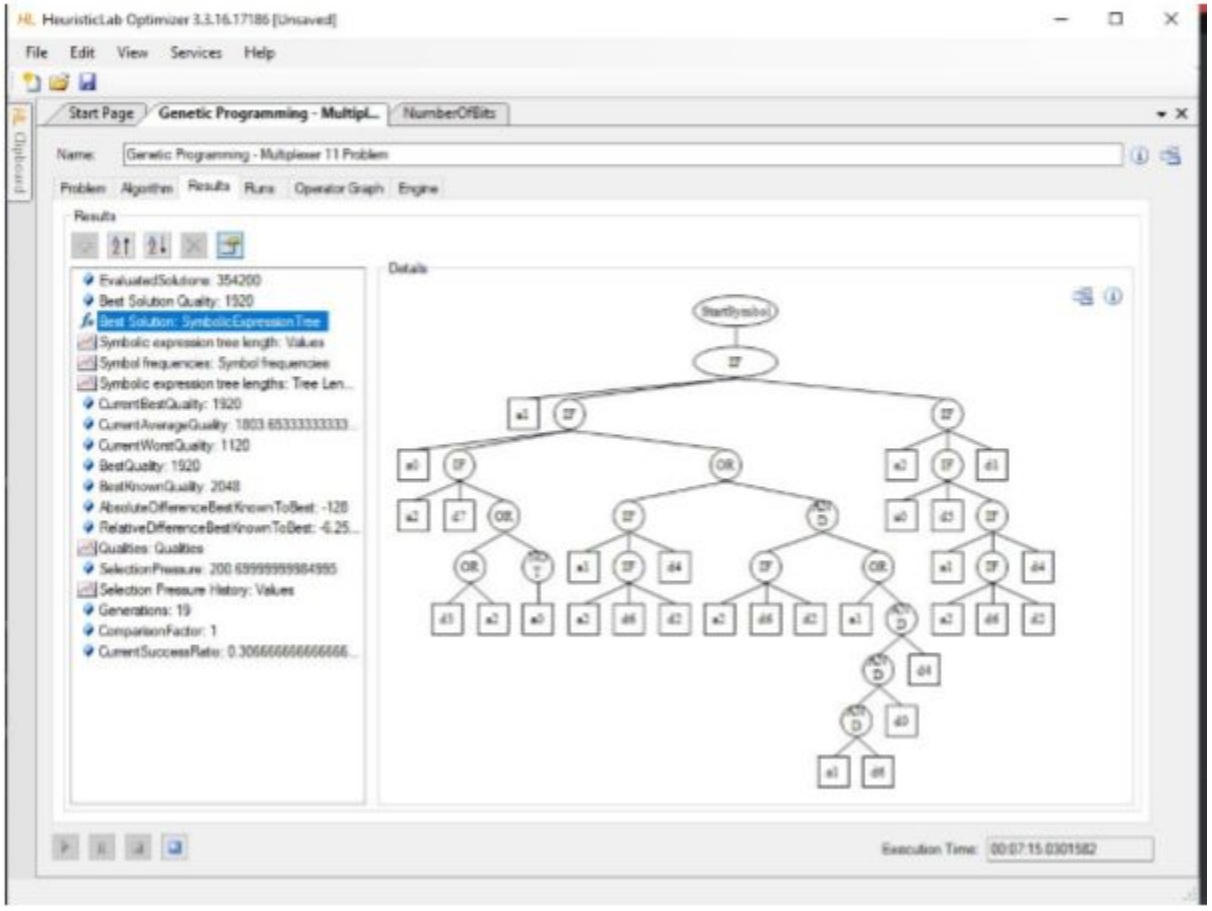


Multiplexor de 11 entradas:

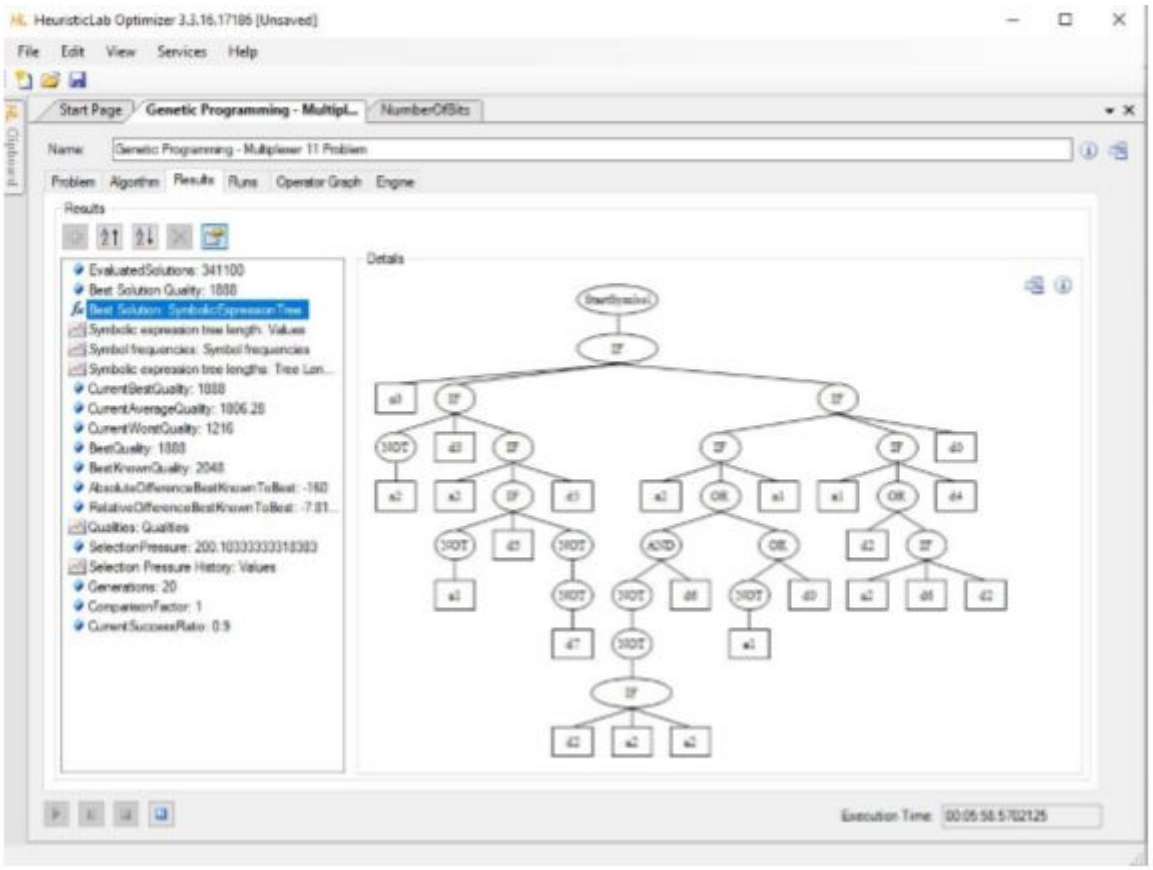
El número máximo de aciertos es 2048. En este primer intento podemos observar que el número de aciertos es 1904, dejando los parámetros por defecto.



Hemos multiplicado en 3 la élite y la población para ver si se acercaba más al número máximo de aciertos. Se acerca algo más pero no demasiado, solo subiendo a 1920 aciertos.



Hemos deducido que cuanto más subamos los parámetros, más cerca estaremos de la mejor solución posible. Aquí tenemos un último intento con 1 de élite y 300 población. Vemos que el número de aciertos disminuye un poco.



Descripción del trabajo realizado

No ha habido una clara distinción de trabajo, debido a que queríamos entender la práctica al completo por ambas partes. Si pudiéramos decir que trabajo ha hecho cada uno, nos dividiremos en quien compartió pantalla esa sesión y quien fue el que programó. Pero ambos estuvimos involucrados.

● Pablo Fernández Jara: Bloating Penalización, mutación terminal simple, funcional simple, de árbol.
● Víctor Gómez-Jareño Guerrero: Bloating Tarpeian, mutación hoist, contracción, expansión y permutación.
● Trabajo conjunto: Clase Árbol y Cromosoma. Clase AGenetico. Memoria y análisis de resultados en Heuristic Lab. Las inicializaciones van incluidas en la clase árbol.