# Memoria Practica 3: Multiplexor con Gramáticas evolutivas

Integrantes: Mario Daniel Gallardo Cruzado

Para mas información sobre la participación se puede acceder a <https://github.com/danisilver/pevolutiva/graphs/contributors>

En los ejemplos mostrados hay capturas con la semilla para reproducir los resultados **(hacer falta presionar enter en la casilla seed, cuando se desactiva la semilla aleatoria también)**

En la pestaña Log se pueden ver los resultados, en esta práctica además se abre una ventana con la representación del árbol, este comportamiento no se puede cambiar.

La parte correspondiente a esta práctica se puede acceder con el desplegable “función” y eligiendo “Multiplexor”, también están la practica 1 y la 2, otro nombre más descriptivo para este desplegable seria “problema”.

En este problema también se puede seleccionar la pestaña Problem View donde se muestran las posibles combinaciones de las entradas y resultados del multiplexor real, cambiar el valor nAddrInputs actualiza esta pestaña **(con nAddrInputs=4 se genera en segundo plano y tarda mucho, a pesar de ser generado con un hilo individual)**

Las clases nuevas son las siguientes:

CruceNodeXchng: Con este cruce intercambiamos nodos entre dos árboles.

MutacionFuncionalSimple: Se cambian las funciones, AND -> OR, OR->AND.

MutacionTerminalSimple: se escoge un terminal y se reemplaza por otro diferente.

MutacionPermutarArgs: se intercambia el orden dos hijos.

MutacionNodeRestart: se escoge un nodo y se inicializa desde cero.

MutacionTreeHoist: se escoge un nodo y se hace ese nodo el árbol raíz.

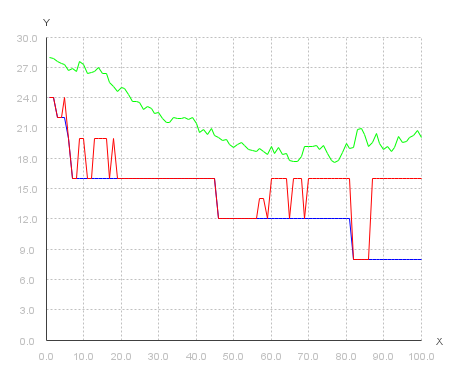
TarpeianBloating: penaliza aquellos individuos con mayor número de nodos basado en una proporción N.

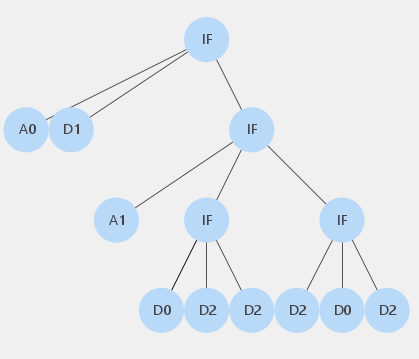
PoliMcPheeBloating: igual que la anterior pero se penaliza conforme a una proporción dependiente del numero de nodos y de fitness.

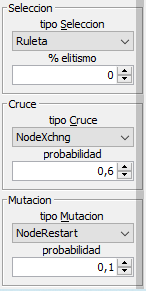
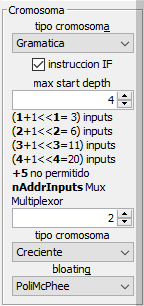
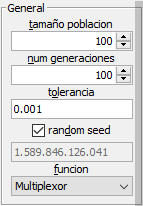
## Parte 1 y 2

A continuación, se muestra algunos resultados:

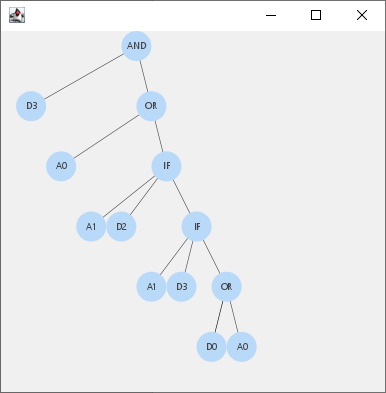
mejor global: evalua:8.0 representación: (IF A0 D1 (IF A1 (IF D0 D2 D2)(IF D2 D0 D2)))

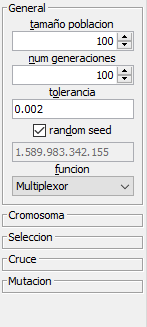
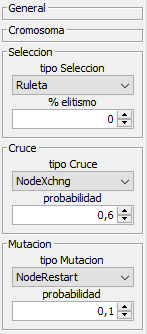
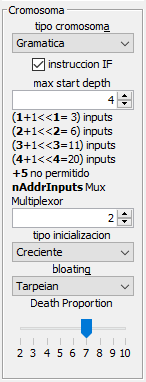


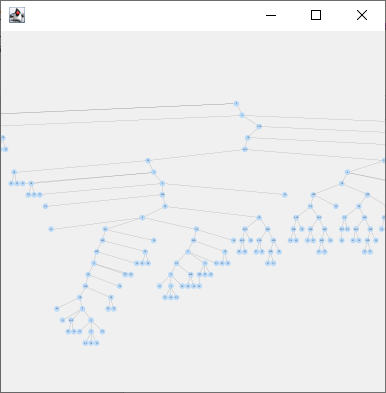


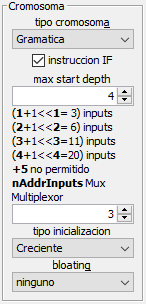
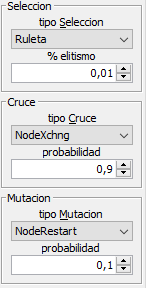


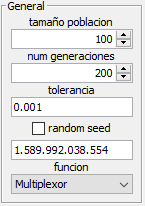
El control del bloating se ha implementado como un adaptador de la Interfaz Selección, de esta manera el bloating se ejecuta previamente a la selección y la selección es la clase envuelta, cualquiera de las disponibles.





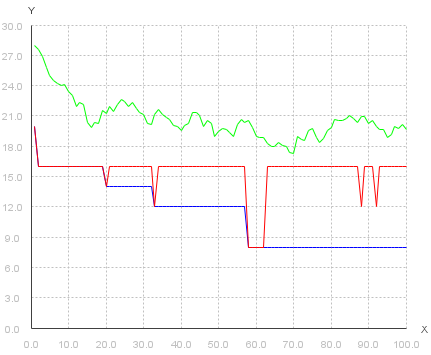
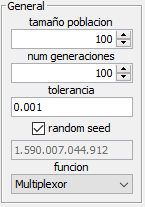
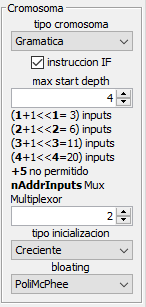
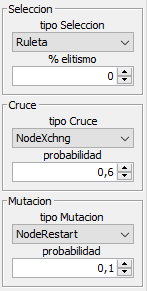


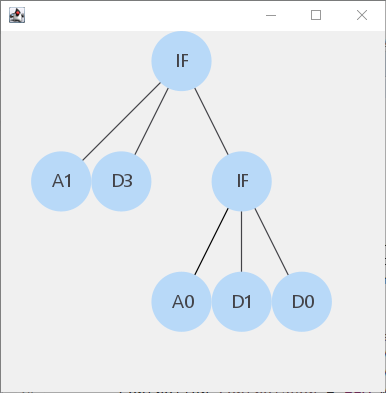


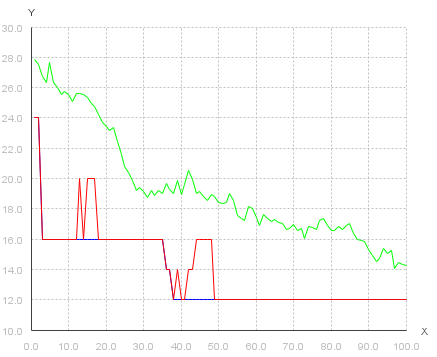
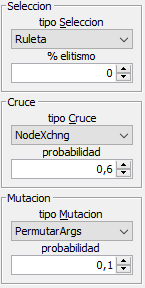
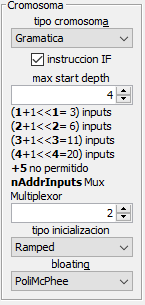


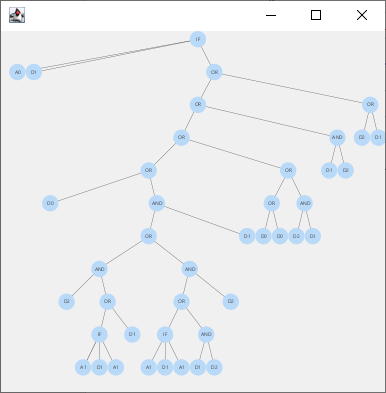
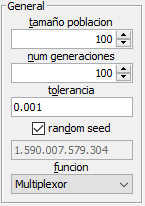
Como vemos el árbol resultante es muy grande (profundidad 26) debido a que sin control del bloating puede haber intrones y el cruce puede generar arboles muy profundos.

mejor global:evalua:640.0 representación:(IF A2 D5 (IF (OR (IF A1 (IF D1 D3 D3 )D0 )(OR D7 D7 ))(AND (OR (AND (OR (OR D2 D7 )(IF A1 (OR D2 D7 )(IF D1 (AND D6 (OR (IF A1 (OR (AND (AND (IF (OR (AND (OR A0 (IF A1 (AND D2 D2 )(IF D1 (IF A2 D4 D1 )D3 )))(OR D2 D7 ))D5 )A1 D3 )D2 )(OR D2 D5 ))D0 )(OR (AND (IF (OR (IF A1 (IF D1 D3 D3 )D0 )(AND D2 D2 ))(IF (NOT A2 )D7 D2 )D1 )(OR D2 D7 ))A0 ))(OR (AND (AND D6 A0 )D2 )(AND (AND A1 D2 )(AND (AND A1 D1 )D5 )))))D3 )))(OR (IF (OR (AND (OR (AND (AND D6 A0 )D2 )(AND (AND A1 D2 )(AND (AND A1 D1 )D5 )))D2 )(AND (OR (OR (NOT A2 )D5 )(AND (AND A1 D2 )(AND (AND A1 D1 )D5 )))(AND (AND A1 D1 )D5 )))D7 D2 )(AND D6 (OR (IF A1 (OR (OR (NOT A2 )D5 )D0 )(OR (IF D1 D3 D3 )A0 ))(IF (OR (NOT A2 )D5 )A1 D3 )))))D5 )(IF A1 (AND D2 D2 )D0 ))D1 ) )



 Con la mutación NodeRestart es mas fácil obtener mejores resultados que con el resto de mutaciones sin embargo no seria mala idea concatenar mutaciones.



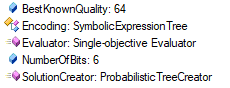
mejor global:evalua:12.0 representación:(IF A0 D1 (OR (OR (OR (OR D0 (AND (OR (AND D2 (OR (IF A1 D1 A1 )D1 ))(AND (OR (IF A1 D1 A1 )(AND D1 D2 ))D2 ))D1 ))(OR (OR D0 D0 )(AND D2 D1 )))(AND D1 D2 ))(OR D2 D1 )) )

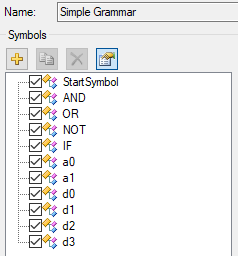
## Parte 3

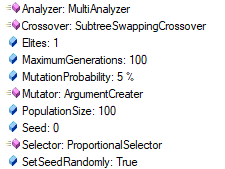
Para resolver el problema con heuristic lab creamos un nuevo problema 

 luego creamos un nuevo problema y elegimos

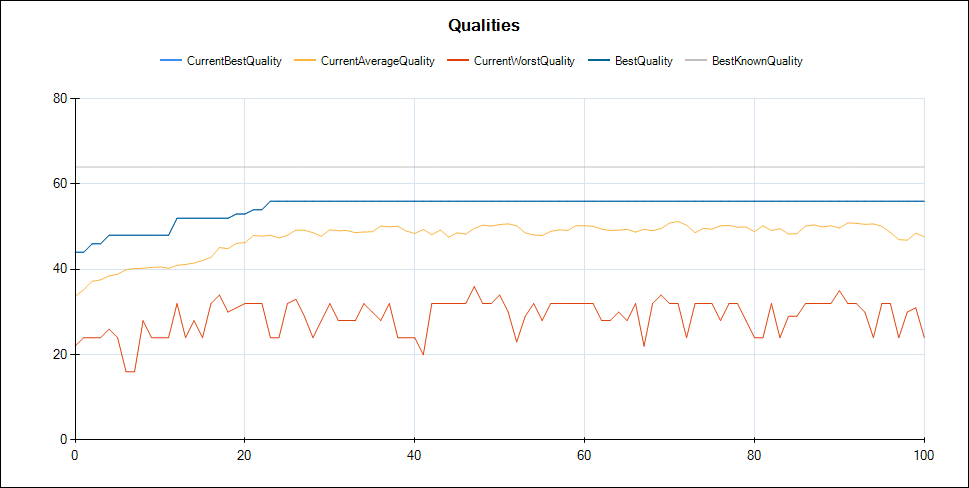


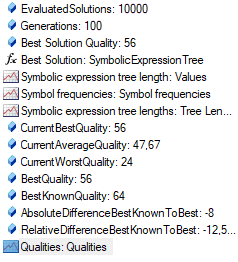
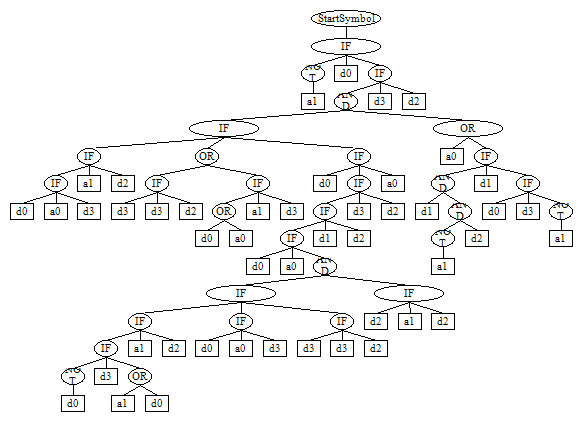
Estos son los parámetros del problema, BestKnowQuality es la mejor solución posible, es decir 64 aciertos, a diferencia de mi enfoque que consiste en reducir fallos, es decir mi mejor solución posible es 0.

Encoding es el equivalente a la clase Arbol, con la que codificamos la gramática, si abrimos encontramos los terminales y las funciones.

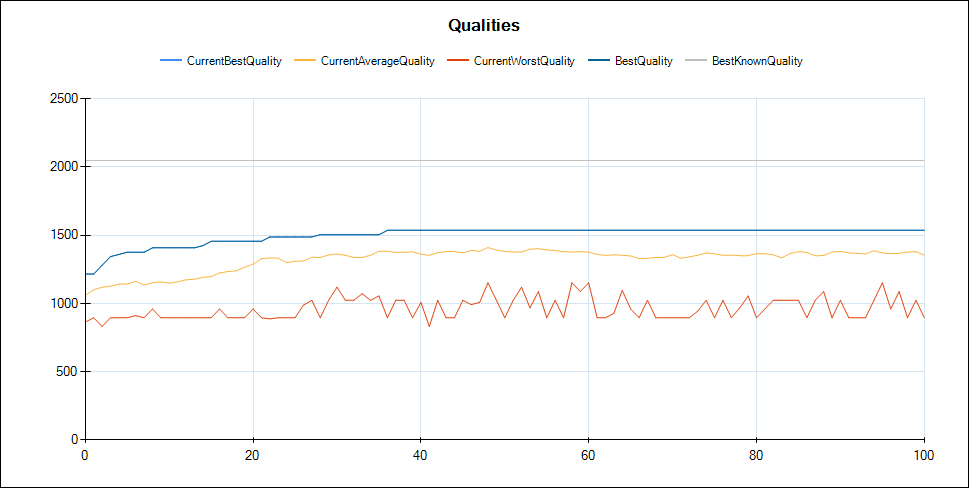
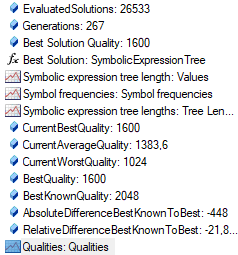
Pasando a la pestaña algoritmo nos encontramos con el mismo tipo de cruce que el nuestro, y la mutación ArgumentCreater que es el equivalente a nuestra mutacion NodeRestart.

Vemos tambien que el metodo de selección es ProportionalSelector o también llamado Selección por ruleta

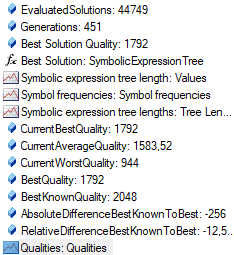
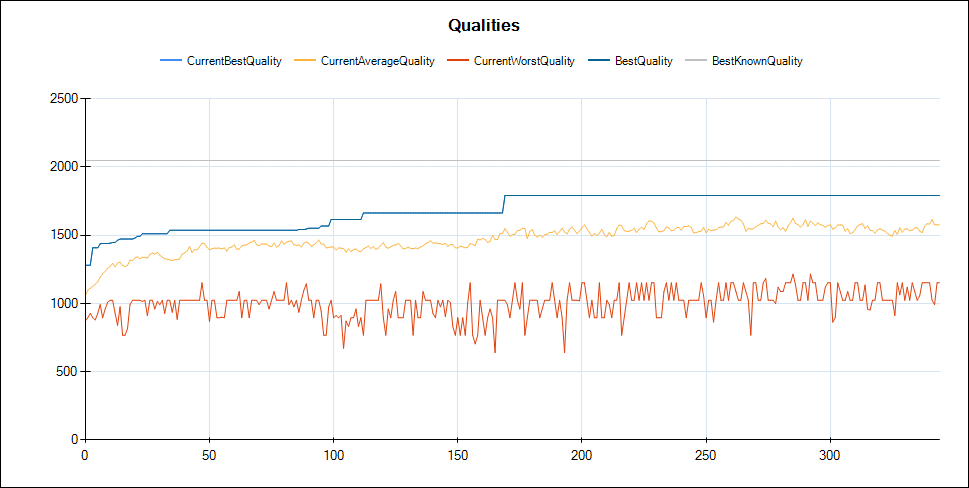
Resultados: 

En esta ejecucion vemos que el resultado es similar al que obteniamos arriba con 8 fallos aquí ha conseguido 56 aciertos que es lo mismo dicho de otra forma.





448 fallos hasta la generacion 267 pero no llega a encontrar mejores soluciones, debido a que espacio de soluciones es muy grande hace falta buscar mejores heuristicas

Se llega a una mejor solucion en la ejecucion de abajo donde se ha utilizado otra mutación: ReplaceBranchManipulation

