PROGRAMACIÓN EVOLUTIVA

Universidad Complutense de Madrid

Ingeniería del Software e Inteligencia Artificial

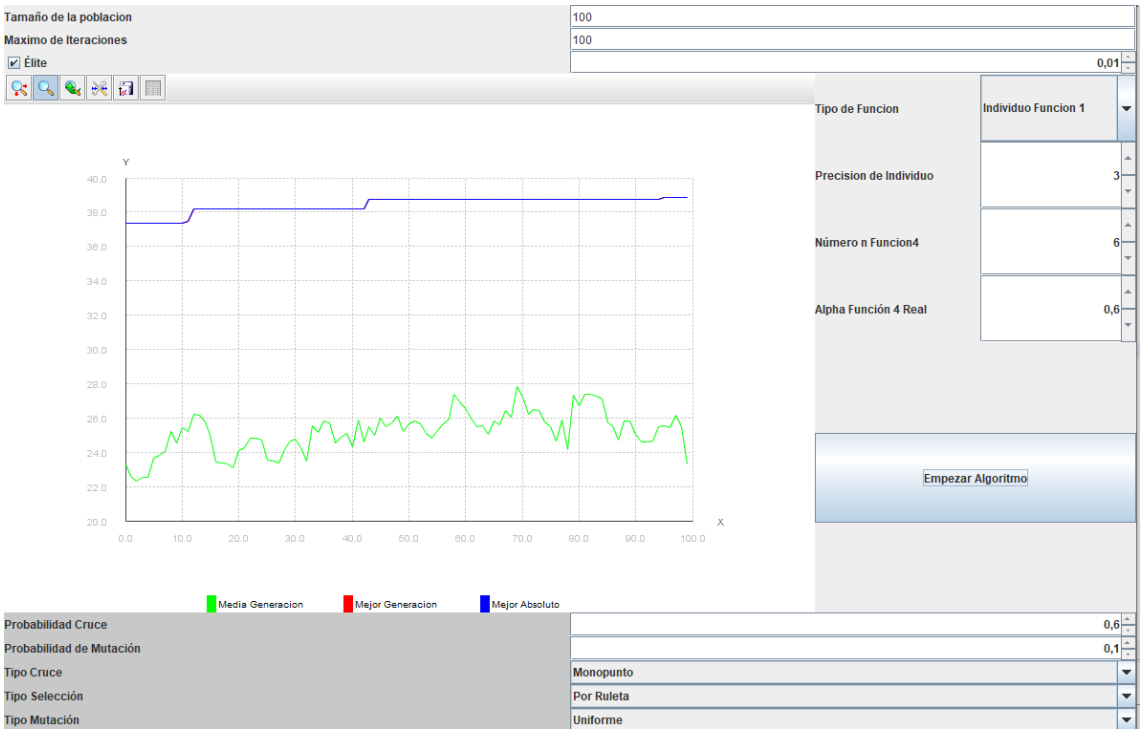
Pablo Villapún Martín

Sandra Mondragón Lázaro

GRÁFICAS POR FUNCIÓN

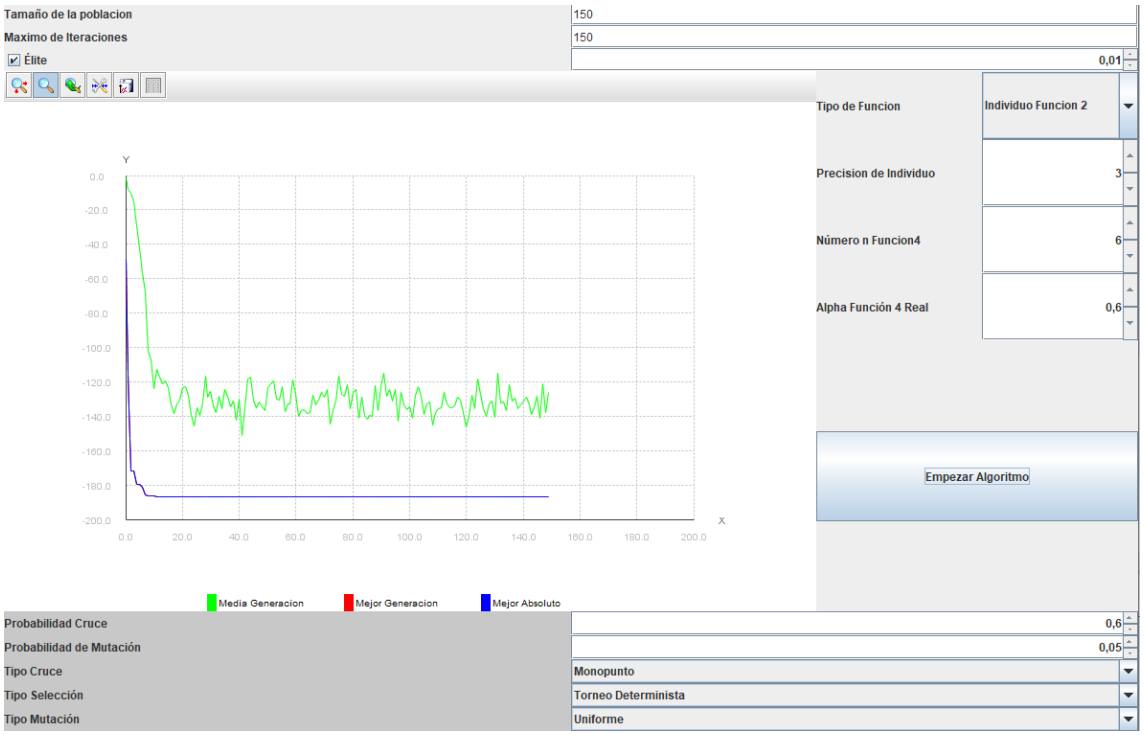
Función1

Valor máximo hallado: `38.84952213298394.`



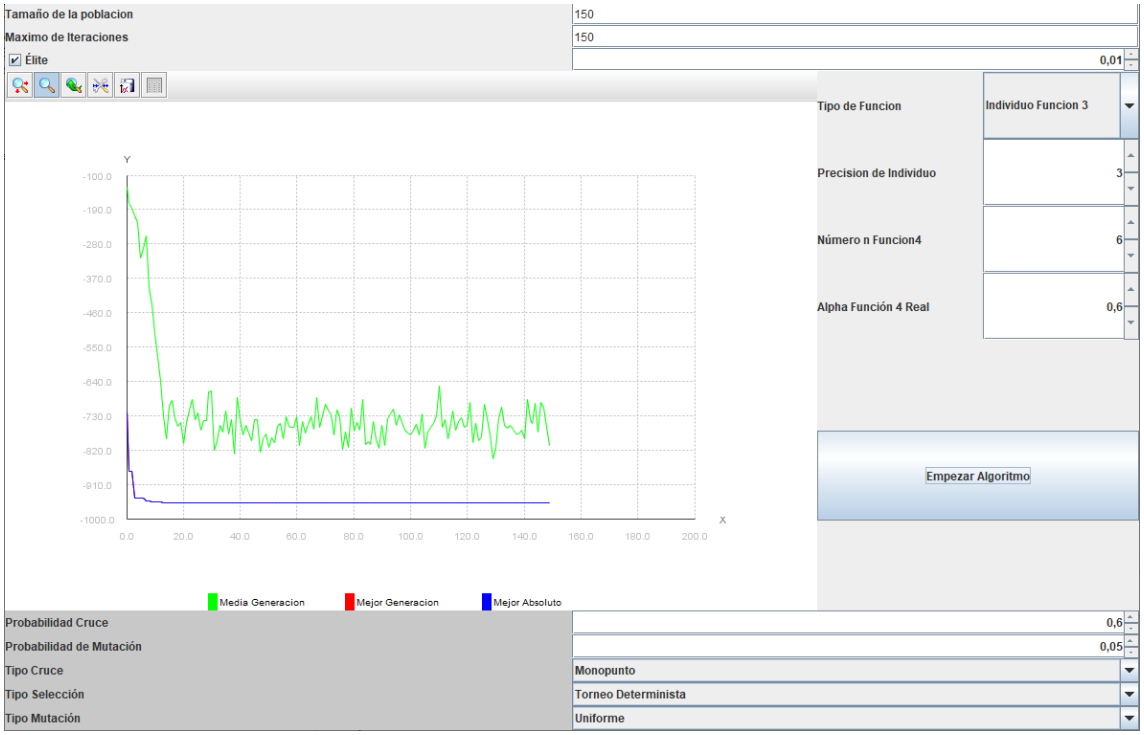
Función2

Valor mínimo hallado: -186.73072206145147



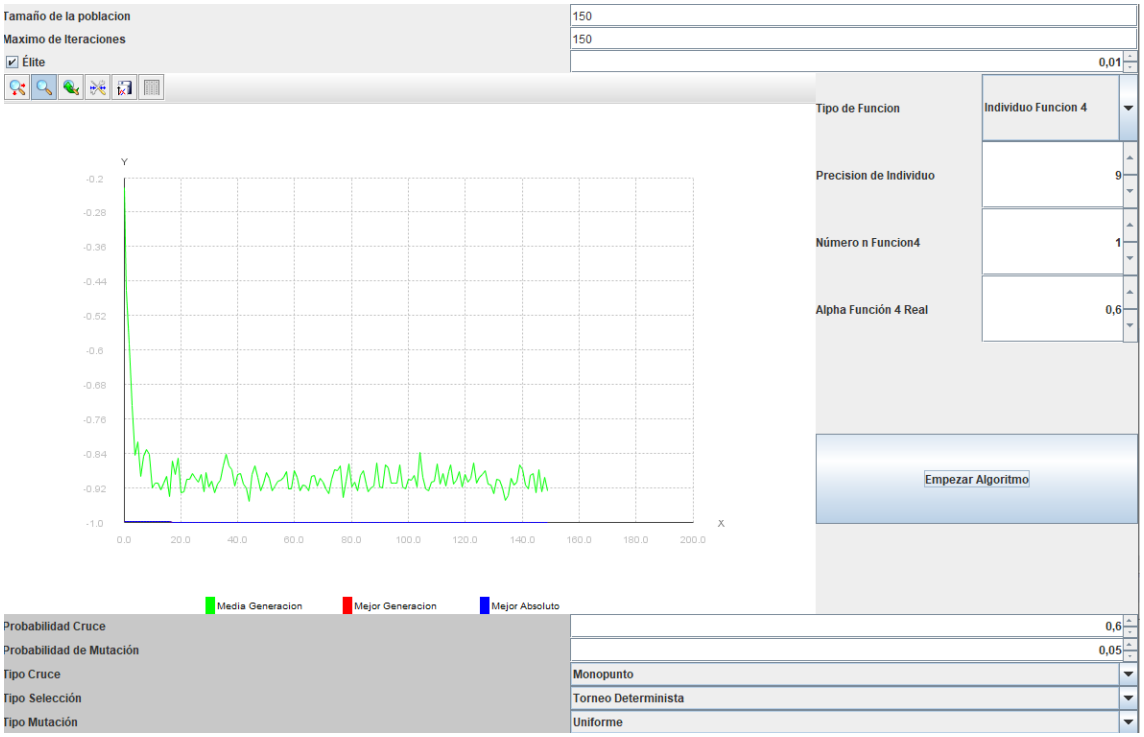
Función3

Valor mínimo hallado: -959.5795635413289



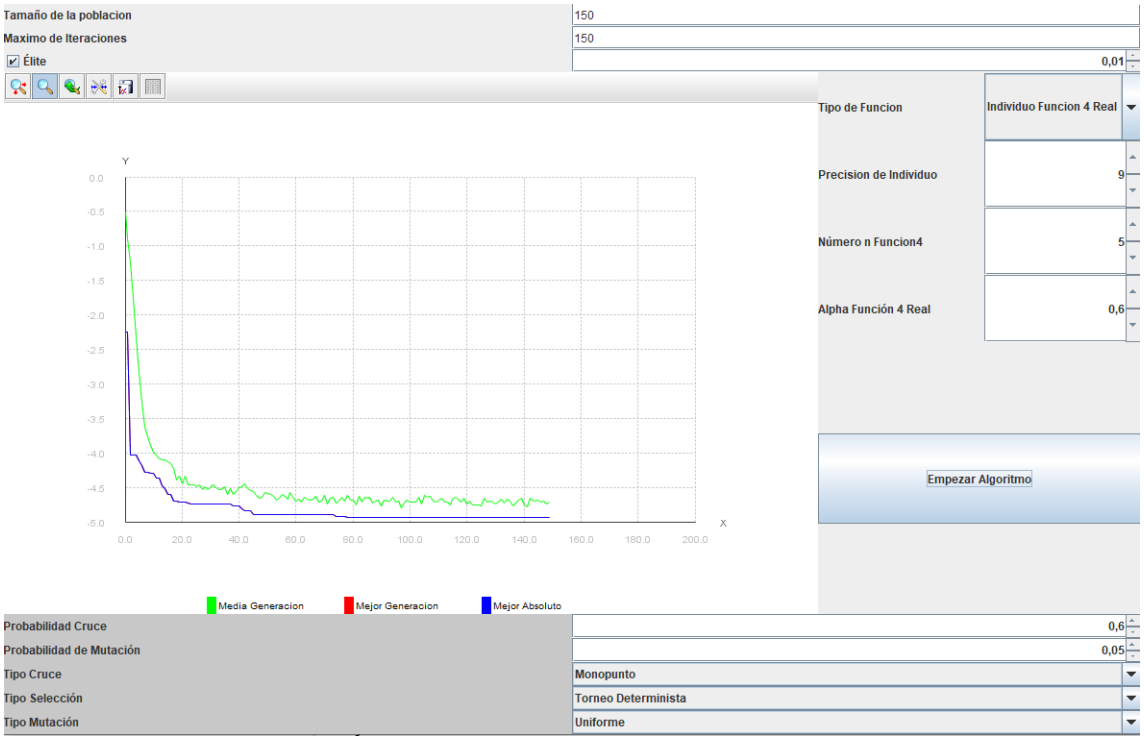
Función4

Valor mínimo hallado para n = 1: -1.0, n = 2: -1.959091269896006,
n = 3: -3.8863408700290196 (misma configuración cambiando n)



Función4 con Reales

Valor mínimo hallado para n = 1: -1.0, n = 2: -1.959091269896006, n = 3: -3.8863408700290196 (misma configuración cambiando n)



CONCLUSIONES DE LA PRÁCTICA

- El elitismo mejora considerablemente los resultados. Con individuos positivos es más fácil de alcanzar el objetivo dado sin elitismo, pero para funciones con individuos negativos ayuda considerablemente.
- El mejor método de selección con el que se comporta de la mejor manera el algoritmo es torneo (tanto determinista como probabilístico) y el peor ruleta (dado que es más al azar).
- Se puede apreciar como la media y el mejor de la generación aumentan o disminuyen dependiendo de si el problema es de maximización o minimización respectivamente. Estos valores van convergiendo conforme aumenta el número de iteraciones, implicando una mejoría de nuestra población.
- El mejor absoluto pasa a ser el mejor de la generación cuando usamos élite.
- Como problema a mencionar, nos dimos cuenta de la importancia de las referencias y las copias de los individuos en la fase de selección, para evitar que en el cruce no se superpusieran.

DETALLES DE IMPLEMENTACIÓN

algoritmoGenetico.cruces

Aquí encontramos todos los cruces dados. Partimos de una clase padre **Cruce** que tiene los métodos de cruzar y **buscarIndividuo** (este último sirve para buscar un individuo que no haya sido cruzado)

algoritmoGenetico.individuos

Aquí encontramos todos los individuos dados. Partimos de una clase genérica **Individuo** que tiene los métodos necesarios. De esta clase parten:

- **IndividuoBoolean** clase que implementa el individuo para el tipo *boolean* y de la que hereda los individuos del 1 al 4
- **IndividuoReal** clase que implementa el individuo para el tipo *double* y de la que hereda el IndividuoFuncion4_Real

Además de esto, encontramos una clase que es la encargada de realizar los individuos según el tipo que nos indiquen.

algoritmoGenetico

Aquí encontramos la clase de **AlgortimoGenético**, encargada del bucle principal del algoritmo. Contiene **métodos para la inicialización y configuración** del algoritmo además de **métodos de evaluación** que guardan la información para luego ser puesta en la **gráfica**.

algoritmoGenetico.mutacion

Aquí encontramos todas las mutaciones dadas. Partimos de una clase padre **Mutacion** que tiene el método de mutar.

algoritmoGenetico.seleccion

Aquí encontramos todas las selecciones dadas. Partimos de una clase padre **Seleccion** que tiene los métodos de seleccionar y **calculaFitness** (este último calcula el fitness de los individuos y los devuelve, en caso de tener negativos, desplazados).

GUI

Aquí encontramos la clase **UIAplication** encargada de generar la ventana de **Jframe** y la interfaz del algoritmo.

REPARTO DE TAREAS

La realización de la base del algoritmo genético, los métodos de selección, cruce y mutación y el IndividuoFuncion1 se han realizado con la técnica de **pair-programming.**

- Implementación de interfaz y clases padre de los individuos ha sido hecho por Sandra Mondragón.
- Implementación del resto de individuos ha sido hecho por Pablo Vilapún.