PROGRAMACIÓN EVOLUTIVA

Universidad Complutense de Madrid

Ingeniería del Software e Inteligencia Artificial



Pablo Villapún Martín

Sandra Mondragón Lázaro

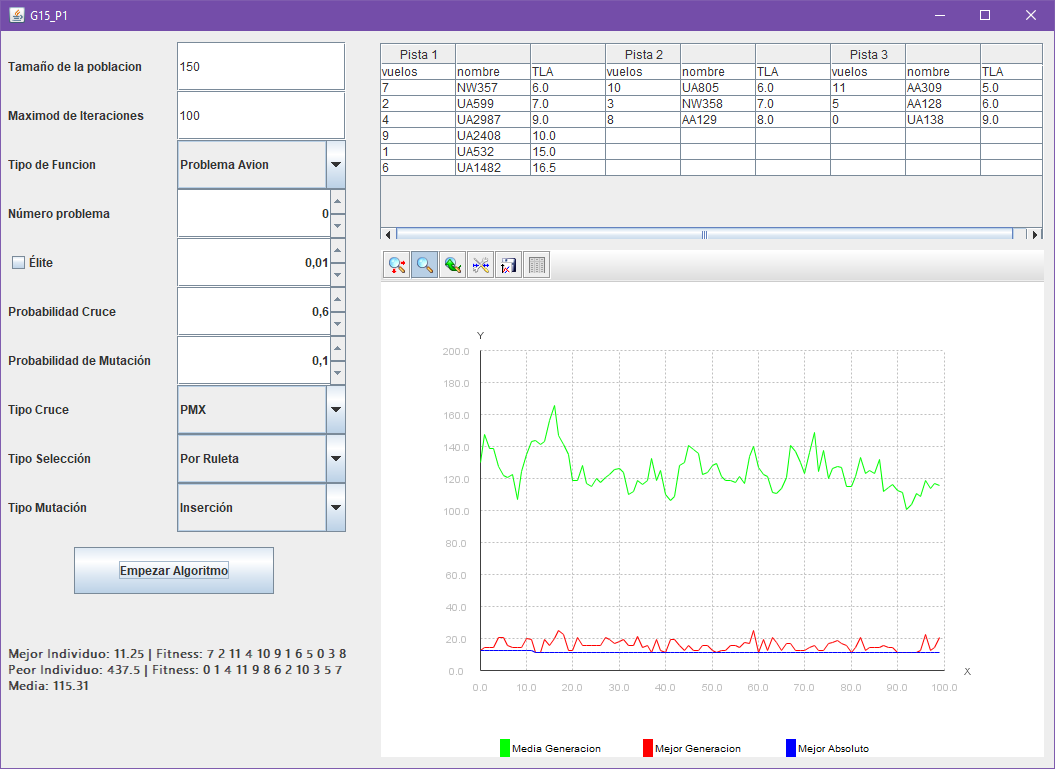
# GRÁFICAS POR PROBLEMA

En esta práctica constamos de tres ejemplos de un mismo problema.

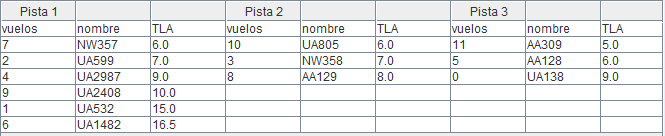
## Problema 1

Este problema es el ofrecido por la práctica, con 3 pistas y 12 vuelos:

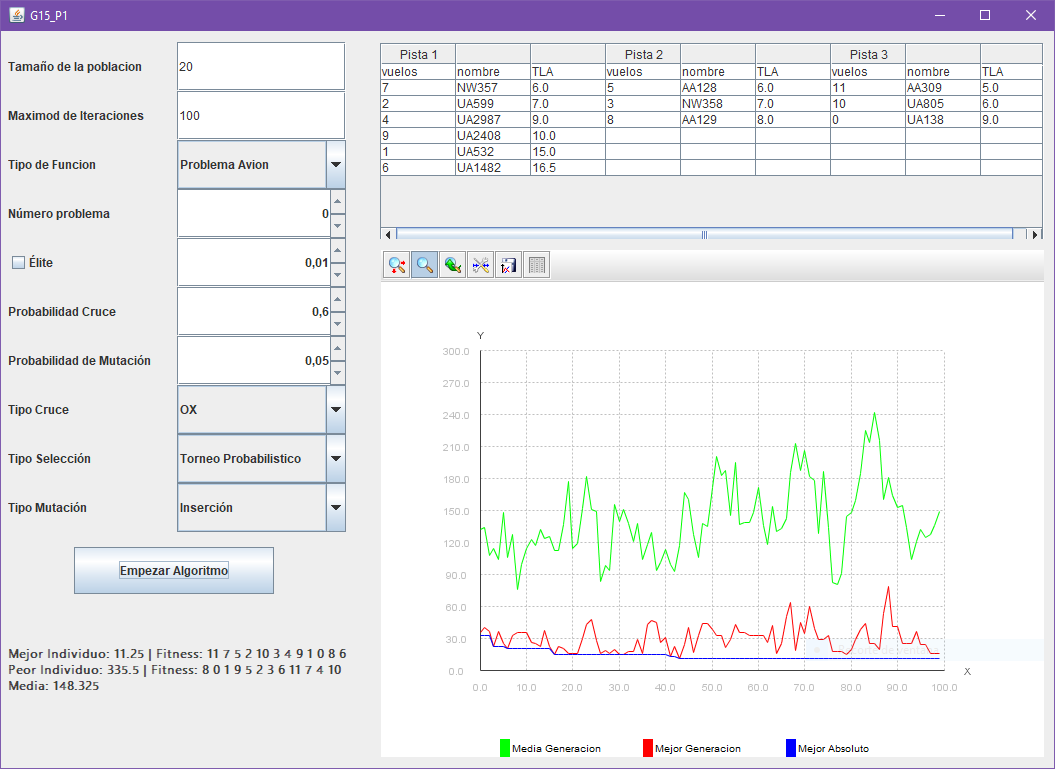
El **mejor individuo** es **11.25**. Al haber muchos individuos y ser un caso simple, llega muy rápido al mejor individuo (y por eso pareciera que la gráfica se queda estancada).

El **peor individuo** es **de 337.5** y la **media** de **143.8.**

El mejor individuo es **7 2 11 4 10 9 1 6 5 0 3 8** donde su colocación en pistas se refleja de la siguiente forma:



En este mismo problema, probando con menos individuos, podemos comprobar que sigue llegando correctamente a 11.25 pero ahora le cuesta más generaciones. Esto lo podemos comprobar en la siguiente imagen:

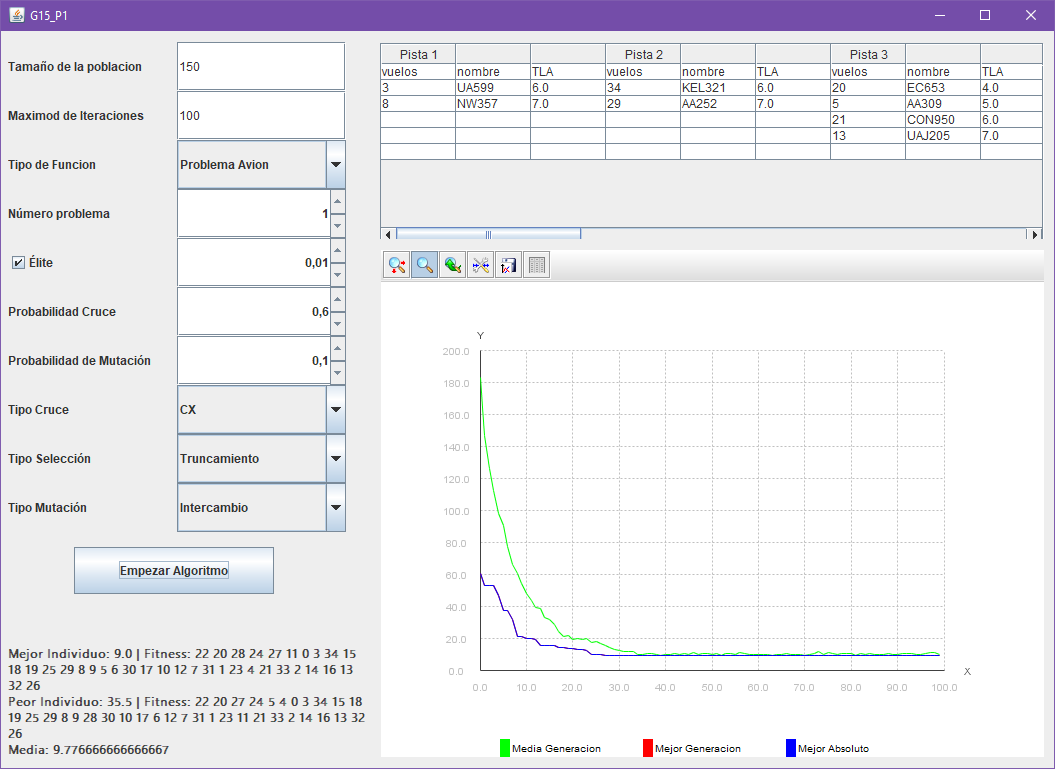


## Problema 2

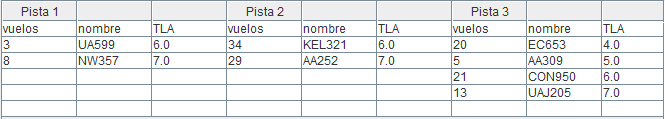
Este problema es inventado, con 10 pistas y 35 vuelos:

El **mejor individuo** es **9.0**. Este caso es más complejo, por lo que **tarda más en ejecutar**. Al ser más complejo también necesita de, o bien **más población**, o bien de **más generaciones** para poder **llegar al mejor individuo**.

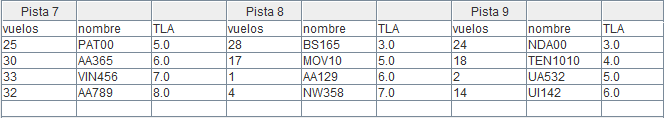
El **peor individuo** (al usar truncamiento y élite) es de **35.5**, pero usando otros métodos de media suele rondar los 84.4.

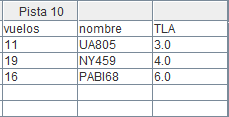
La **media** es de 9.776 en este caso, pero de normal (sin elite ni truncamiento) **suele rondar** en **130 ó150.**

El mejor individuo es **22 20 28 24 27 11 0 3 34 15 18 19 25 19 8 9 5 6 30 17 10 12 7 31 1 23 4 21 33 2 14 16 13 32 26**, donde su colocación en pistas se refleja de la siguiente forma:





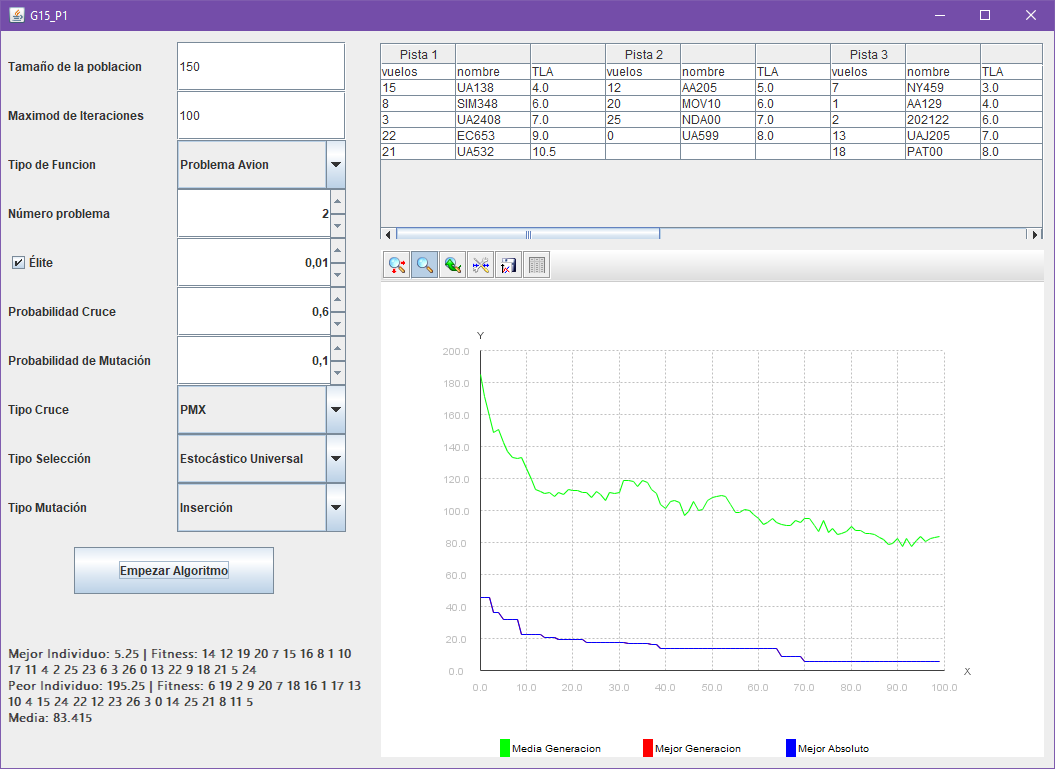




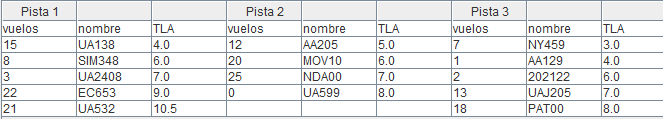
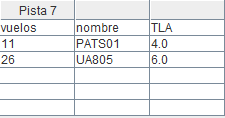
## Problema 3

Este problema es inventado, con 7 pistas y 27 vuelos:

El **mejor individuo** es **5.25**. Este caso es más complejo, por lo que **tarda más en ejecutar**. Al ser más complejo también necesita de, o bien **más población**, o bien de **más generaciones** para poder **llegar al mejor individuo**.

El **peor individuo** es de **195.25** y la **media** es de **83.47.**

El mejor individuo es **14 12 19 10 7 15 16 8 1 10 17 11 4 2 25 23 6 3 26 0 13 22 9 18 21 5 24** donde su colocación en pistas se refleja de la siguiente forma:



# CONCLUSIONES DE LA PRÁCTICA

* Dado que los valores del cromosoma no pueden repetirse entre sí, el resultado para problemas en los que el número de pistas y aviones son bajos la solución se encuentra mucho más prematuramente, ya que la combinación óptima tiene más posibilidades de aparecer al iniciar de forma aleatoria los individuos.
* La mutación heurística, a pesar de ser la más costosa en ejecución ya que tiene que realizar y comprobar todas las permutaciones de los valores seleccionados, incrementa la posibilidad de obtener mejores individuos ya que muta buscando la combinación óptima.
* Los métodos de selección que se quedan con los mejores individuos como truncamiento o torneo determinista hacen que la media tienda hacia valores más bajos ya que los individuos menos favorables son menos seleccionados.
* La probabilidad de mutación se debe incrementar para reflejarse mejor en la gráfica de evolución, ya que a diferencia de la práctica 1 donde dicha probabilidad determinaba si un solo gen mutaba, ahora sirve para determinar si un sólo individuo cambia, por lo que es razonable aumentar un poco su valor respecto a la anterior práctica.
* Existen varias posibles soluciones a cada problema, ya que hay óptimos representados por combinaciones parecidas pero que difieren en algún valor.
* Con problemas grandes es necesario de más generaciones (o de una mayor población) para que llegue al mejor individuo.

# 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 el **IndividuoAvion.** Este individuo es el que codifica la ordenación de aviones a pista. En su método **getValor** se calcula de forma óptima el posicionamiento de los aviones. El **método** para **calcular el fitness** es con el **TEL más bajo entre todas las pistas.**

## algoritmoGenetico.aviones

Aquí encontramos las clases donde se guarda información de los problemas. **InfoAvion** guarda la información de un vuelo (su peso y su nombre); **InfoPista** guarda el vuelo y el TLA asignado a ese vuelo (se utiliza para el cálculo del fitness); **TráficoAereo** es un **Singleton** que contiene las matrices con los datos de los distintos problemas (TEL, SEP y un array de InfoAvion).

La clase **GeneraTablaAvion** es una clase que sirve para abstraer la creación de la tabla de la interfaz gráfica, añadiendo columnas dada una información obtenida sobre el resultado final.

## 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**. Aquí también se realiza la **tabla de aviones-pistas.**

## 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

Toda la parte del cálculo del fitness del individuo y de la información común (paquete aviones) fue hecho utilizando **pair-programming.** También el método de selección por ranking.

* **Sandra Mondragón**: Cruce CO, cruce CX, cruce OX, mutación heurística y mutación por inserción y cambios en la interfaz.
* **Pablo Villapún**: Cruce OXPP, cruce OXOP, cruce PMX, mutación intercambio y mutación inversión.