Programación Evolutiva

Memoria de la práctica 2

David Alfonso Starry González

Daniel Pizarro Gallego

2024

Índice:

1. Introducción

El problema que vamos a optimizar consiste en un Aeropuerto con *n* aviones y *m* pistas (*n*>*m*).

Tenemos que asignar a los *n* aviones una pista, y reducir el tiempo de espera en el aire de cada avión. Es un problema de **optimización combinatoria**, por lo que el orden de los aviones importa.

Como datos de entrada tenemos:

- TEL: Matriz de tiempos estimados de llegada de cada avión *(n columnas)* a cada una de las pistas *(m filas)*. Estos datos se encuentran en **TELX.txt** (siendo X: el tipo de Aeropuerto)
- SEP: Matriz de tiempos de separación. Cada tiempo depende del tipo de avión que llega a la pista y el tipo de avión que se va para dejarla libre.. Hay 3 tipos de aviones:
  - Pesado (W)
  - Grande (G)
  - Pequeño (P)
- vuelos: array con los identificadores y tipo de cada avión. Estos datos se encuentran en **vuelosX.txt** (siendo X: el tipo de Aeropuerto)

2. Arquitectura de la aplicación

La aplicación se aplica modelo vista controlador. Al ejecutar el programa, con la clase **Main** se inicializa la interfaz usando la clase **MainWindow**, que extiende a JFrame, y añade el controlador.

El controlador es la clase **ControlPanel** que extiende a JPanel, para crear 2 paneles dentro del principal.

- En el panel izquierdo se introducen los valores para las variables, mediante componentes de JSwing, como JButton, JTextField o JComboBox.
- El panel derecho imprime el gráfico 2D cuando termina una ejecución. Este gráfico tiene 2 variables. El eje x es el número de generaciones, y el eje y el valor fitness, de 1. el mejor absoluto (azul), 2. el mejor de la generación (rojo) y 3. la media de la generación (verde).

Una vez pulsado el botón de ejecutar, se ejecuta el algoritmo genético usando la clase **AlgoritmoGenetico**. En esta clase se guardan los valores almacenados en la interfaz, mediante una clase que almacena los datos, llamada **Valores**.

Los individuos de la población los implementamos con la clase **Individuo**, en el cual usamos la clase **Gen**, que consiste en un array de enteros, en el cual guardamos el orden de los aviones.

La función que calcula el fitness (o adaptación) de cada individuo, se implementa en la clase **Funcion**. El cálculo se implementa de la siguiente manera:

```
fitness = 0;
para cada vuelo del individuo hacer

    //calculamos el TLA del vuelo a cada pista
    TLA = máximo (TLA(v_anterior) + sep(v_anterior, v_actual), TEL);
    // se asigna el vuelo actual a la pista con mínimo TLA (menor_TLA)

   fitness = fitness + (menor_TLA - menor_TEL)²
   //menor_TEL: menor TEL de ese vuelo en todas las pistas.
```

Los métodos de selección se implementan en la clase **Seleccion**. Se aplican igual para los dos individuos. Estos son:

- Ruleta: Selección aleatoria con la probabilidad acumulada de su fitness.
- Torneo determinístico: Se eligen 'k' individuos de la población de forma aleatoria y se elige el mejor. Este proceso se repite hasta llenar la población.
- Torneo probabilístico: Igual que el anterior, pero se elige peor o mejor (aleatoriamente)
- Estocástico universal: Similar al muestreo proporcional pero ahora se genera un único número aleatorio simple r y a partir de él se calculan los restantes. Los individuos se mapean en segmentos contiguos cuyo tamaño es el de su aptitud. a es un número aleatorio entre 0 y 1/tam_seleccionado. Se generan tam_seleccionados puntos en el segmento, y se eligen con sus aptitudes (probabilidades acumuladas). 2 métodos iguales, solo cambia la fórmula para calcular los segmentos:
  - Método1: aj=a+(j-1)/N
  - Método2: aj=(a+j-1)/N
- Truncamiento: Se ordenan por fitness y con el porcentaje 'trunc' se eligen los mejores, 1/trunc veces.
- Restos: Las probabilidades acumuladas se multiplican por 'k', y se seleccionan este número redondeado para abajo veces, y los que falten con otro método.
- **Ranking:** Se basa en el ranking según la ordenación de los individuos por fitness decreciente. El valor asignado a cada individuo depende sólo de su posición en el ranking y no en su valor objetivo. Se calcula con la siguiente fórmula: $p(i) = \frac{1}{n}\left[\beta - 2(\beta.1)\frac{i-1}{n-1}\right], 1 \leq \beta \leq 2.$ β se puede interpretar como la tasa de muestreo del mejor individuo o **Presión selectiva**.

Los métodos de cruce se implementan en la clase **Cruce**. Son los siguientes:

- PMX: Consiste en elegir un tramo de uno de los progenitores y cruzar preservando el orden y la posición de la mayor cantidad posible de ciudades del otro. Algoritmo:
  - Elegir aleatoriamente dos puntos de corte. Intercambiar las dos subcadenas comprendidas entre dichos puntos.
  - Para los valores que faltan en los hijos se copian los valores de los padres:
    - Si un valor no está en la subcadena intercambiada, se copia igual.
    - Si está en la subcadena intercambiada, entonces se sustituye por el valor que tenga dicha subcadena en el otro padre. Si este valor ya está en el hijo se repite este proceso hasta llegar a uno valor que no esté en hijo.

- Cruce por Orden (OX): El cruce por orden consiste en copiar en cada uno de los hijos una subcadena de uno de los padres mientras se mantiene el orden relativo de las ciudades que aparecen en el otro padre. Algoritmo:
  - Elegir aleatoriamente dos puntos de corte. Intercambiar las dos subcadenas comprendidas entre dichos puntos.
  - Para los valores que faltan en los hijos se copian los valores de los padres comenzando a partir de la zona copiada y respetando el orden:
    - Si un valor no está en la subcadena intercambiada, se copia igual.
    - Si está en la subcadena intercambiada, entonces se pasa al siguiente posible.

- OX posiciones prioritarias: Igual que el anterior, pero en vez de copiar un tramo se eligen puntos aleatorios. Algoritmo:
  - Seleccionar puntos aleatorios. Se guardan los valores de los puntos aleatorios del padre1. En el hijo1 se copian del padre2 los valores que no sean los elegidos.
  - Los huecos se rellenan de izquierda a derecha con los valores aleatorios del padre1. Se repite el proceso para el hijo 2, empezando con el padre2.

- CX: Se opera completando "ciclos de sucesión". La descendencia se construye combinando las rutas que interconectan las ciudades de los progenitores. Se construye la tabla de conectividades entre ciudades de uno u otro progenitor. Algoritmo
  - Se crea la tabla de conectividades comprobando los vecinos de cada ciudad en los padres. En la columna i-ésima se añaden las ciudades vecinas de la ciudad *i* de ambos padres.
  - Con la tabla ya formada, para cada hijo se empieza con el primer valor de un padre. Comienza un bucle con un valor actual y de este valor se usa su columna, escogiendo la ciudad que esté menos conectada, es decir, la que menos ciudades tenga en su columna.

- CO: Se ordenan todas las ciudades en una lista dinámica de referencia según cierto criterio. Para construir un individuo se van sacando una a una las ciudades recorridas, codificando en el j-ésimo gen del individuo la posición que tiene la j-ésima ciudad en la lista dinámica. Algoritmo:
    - Tenemos una lista dinámica con números de 1 a *n*, con cada padre se recorre todo el gen de izquierda a derecha.
        - En cada iteración se elimina de la lista el valor i-esimo, así reduciendo la lista y almacenando en un array temporal la posición que ocupaba antes de ser eliminado.
    - Se elige un punto de corte aleatorio y se intercambian los arrays temporales. Con la lista dinámica se vuelve hacer lo mismo. Ahora recorremos los arrays temporales eliminando de la lista dinámica la posición del array temporal, y almacenando en el hijo el valor de la lista dinámica antes de eliminarlo.

- Método propio: En los hijos se guardan las posiciones pares de un padre, y las posiciones impares se rellenan, por orden, con los valores que no estén del otro padre.

Los métodos de mutación se implementan en la clase **Mutacion**.

- Inserción: Se inserta una o varias ciudades elegidas al azar en unas posiciones también elegidas al azar. Al insertar el valor se elimina de su posición antigua y se desplazan las ciudades que estén a la derecha de la posición en la que se inserta.

- Intercambio: Se seleccionan dos puntos al azar y se intercambian.

- Inversión: Consiste en seleccionar dos puntos del individuo al azar e invertir los elementos que hay entre dichos puntos.

- Heurística: En este tipo de mutación se seleccionan n ciudades al azar. A continuación, se generan todas las permutaciones de las ciudades seleccionadas. De todos los individuos que se generan con dichas permutaciones se selecciona el mejor.

- Método propio: Consiste en seleccionar dos puntos del individuo al azar, los valores que hay entre dichos puntos se recolocan de manera aleatoria. Se guardan los valores en un conjunto y se elige aleatoriamente, con la misma probabilidad, un valor, y se almacena en el individuo, eliminando lo del conjunto.

3. Mejoras implementadas

Se aplica un **algoritmo de divide** y vencerás $O(log_2 N)$ para reducir el tiempo de ejecución a la hora de elegir los valores como en ruleta y estocástica universal.

El **Elitismo** consiste en asegurar la supervivencia de un grupo con los mejores individuos de la población. En el controlador se puede especificar un porcentaje entero para el conjunto elitista que sobrevive en cada generación. Este conjunto se calcula en la etapa de evaluación, y se comparan los individuos con su fitness, almacenando los 'r' mejores en una cola de prioridad de mínimos. Así reducimos la complejidad a $O(N\ log_2 R)$ en el caso peor, siendo N el tamaño de población y R el conjunto de élite. Se compara el mínimo valor de los mejores con cada individuo, si el menor de la cola es peor que el individuo actual, se elimina de la cola y se introduce el nuevo, subiendo hasta su posición en la cola.

La **Presión Selectiva** es la mayor o menor tendencia a favorecer a los individuos más aptos. Cuantifica el número esperado de descendientes que se da al miembro más apto de la población**.** $\boldsymbol{PresSel[i] = TamPob \cdot ProbMax[i]}$. ProbMax[i] es la probabilidad de selección mayor en la generación i.

Desplazamiento de la aptitud: Tiene como finalidad hacer que la función de aptitud devuelva valores positivos. Se calcula con la siguiente fórmula: $f[i] = fmax - g[i]$. f[i] es el nuevo fitness del individuo i-ésimo, g[i] es el fitness antiguo. En lugar de utilizar fmax conviene utilizar valores ligeramente mayores (por ejemplo, un 105%) para prevenir que la adaptación revisada se haga nula.

Escalado de la adaptación: Permite controlar la diversidad de las aptitudes. Permite establecer una separación entre los valores de adaptación de distintos individuos que sea apropiada para el funcionamiento de la selección.

Implementamos **escalado Lineal**. Este tipo de escalado puede producir valores de adaptación negativos, que tenemos que evitar, pero en este problema no ocurre. El fitness nuevo se calcula de la siguiente manera $f(x_i)$.

$$f(x_i) = a * g(x_i) + b \qquad a = \frac{(P-1)*\bar{g}}{g_{max}*\bar{g}} \qquad b = (1-a)*\bar{g}$$

Al final no lo usamos porque cambia a peor los resultados obtenidos. Seguramente lo implementamos mal.

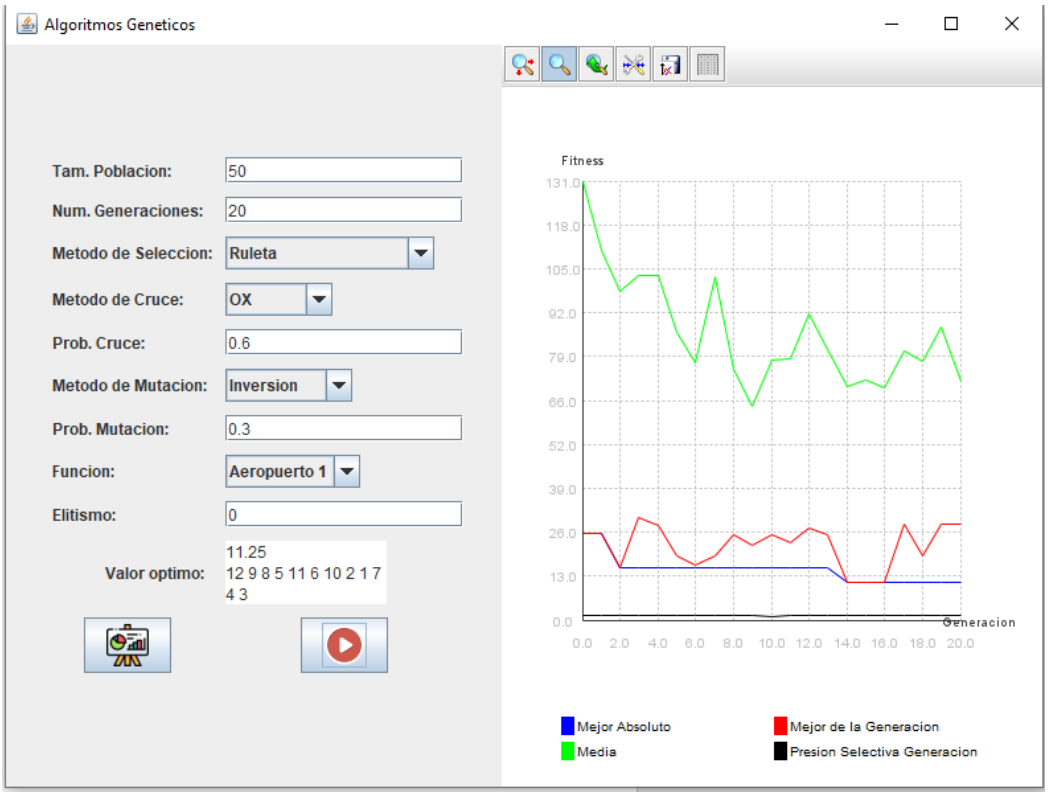
Aeropuerto Extra: Hemos implementado un aeropuerto extra con 100 vuelos y 10 pistas. Los datos de entrada están en la carpeta /data del proyecto java.

Reinicio de población, cada 250 generaciones se reinicia la población, guardando el 10% de la población con mayor aptitud.

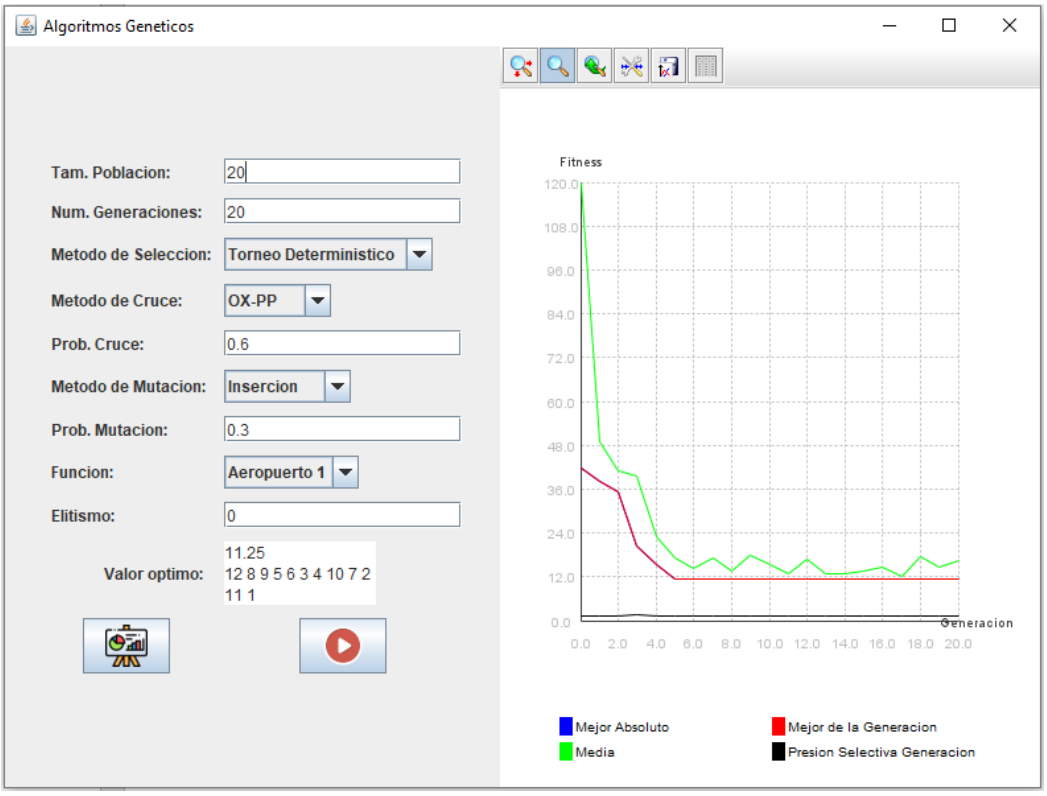
4. Ejecuciones, Análisis y Conclusión

4.1. Ejecuciones

4.1.1. Aeropuerto 1

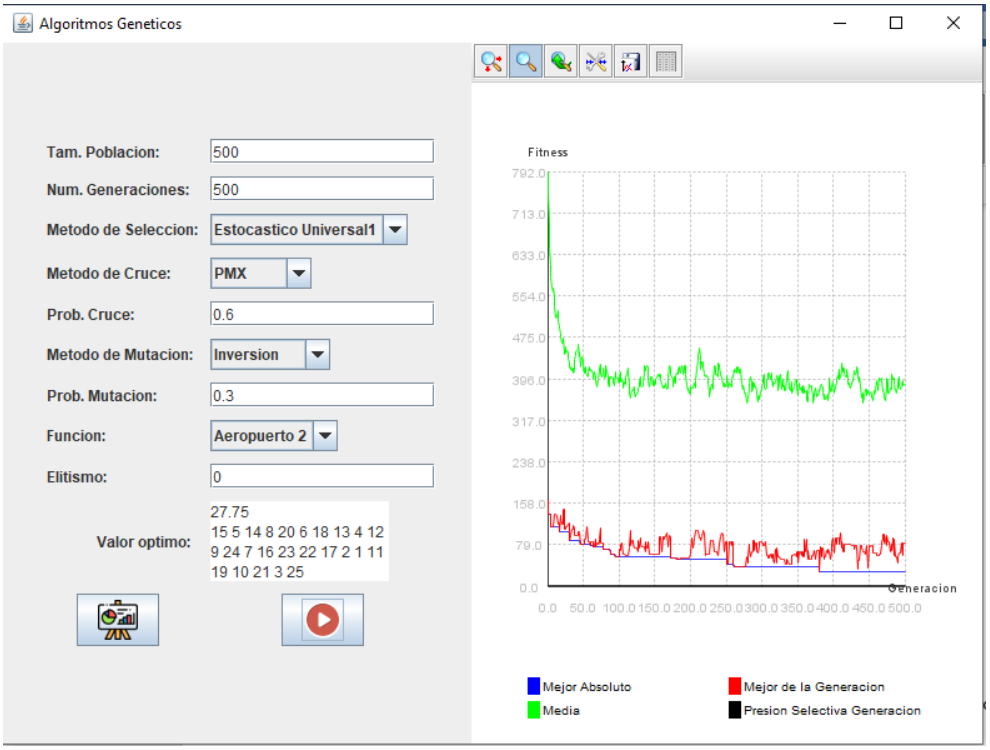


Con los peores métodos de selección, cruce y mutación logra obtener el valor óptimo en pocas generaciones de manera repetida.

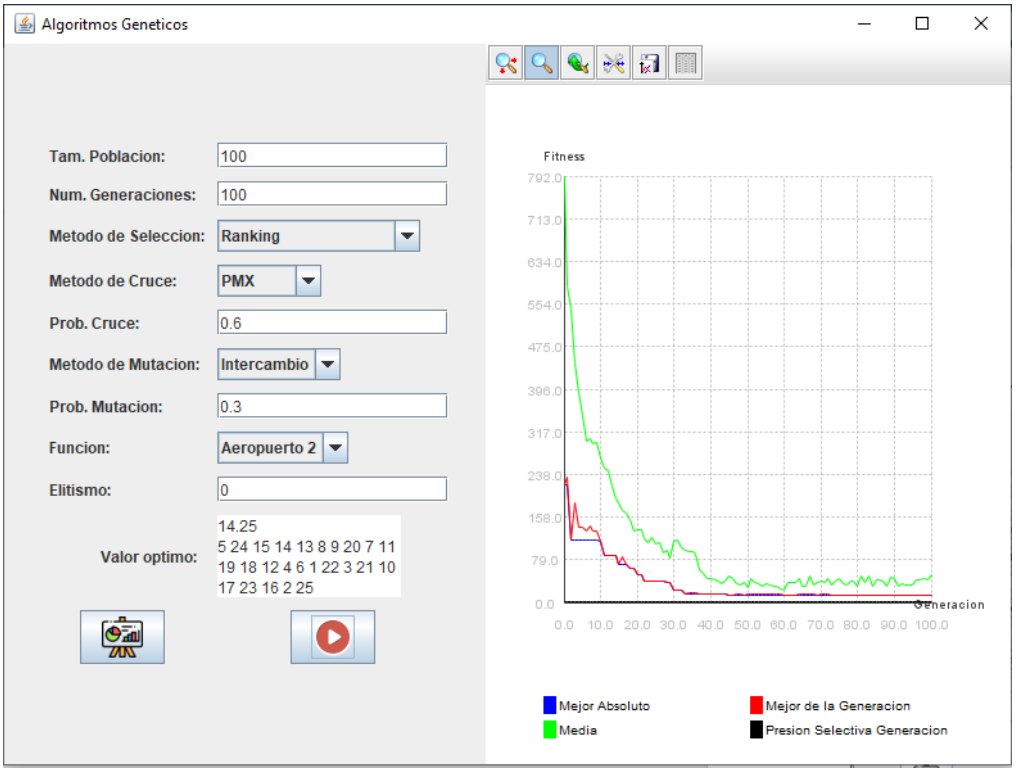


Con los métodos más eficientes que hemos obtenido mediante pruebas, obtiene el valor óptimo pocas generaciones y con poca población.

4.1.2. Aeropuerto 2

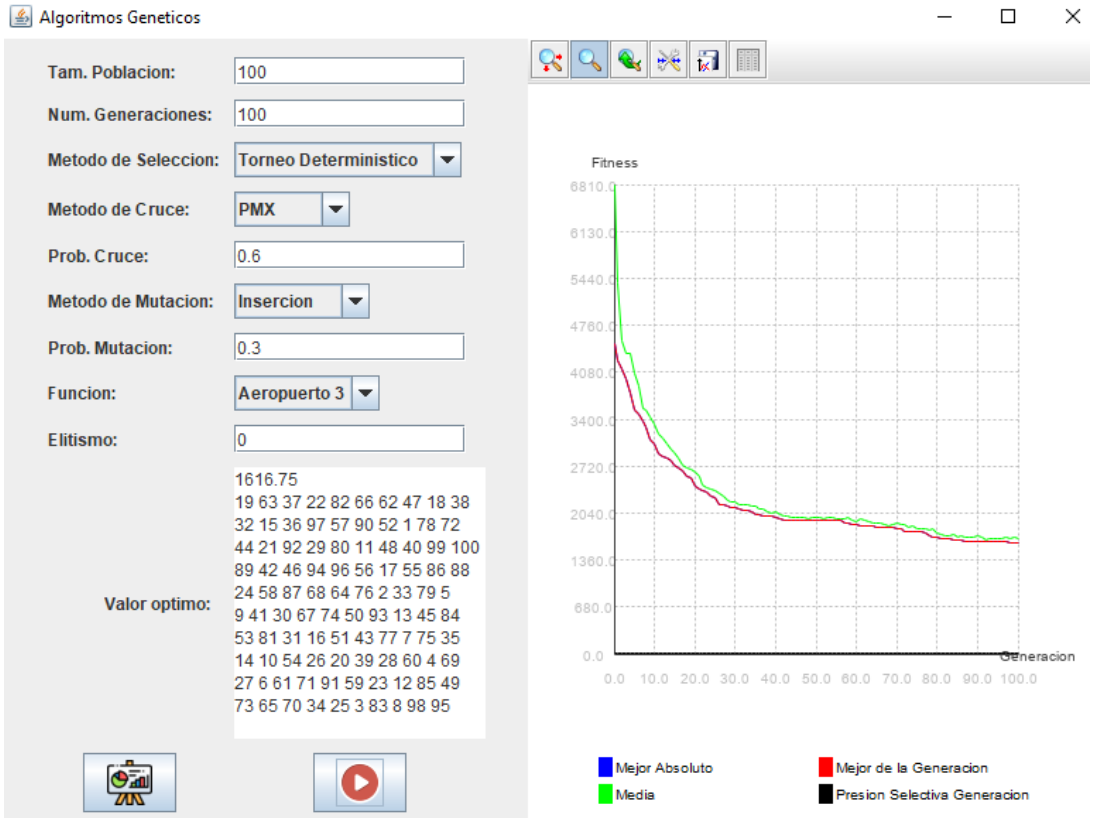


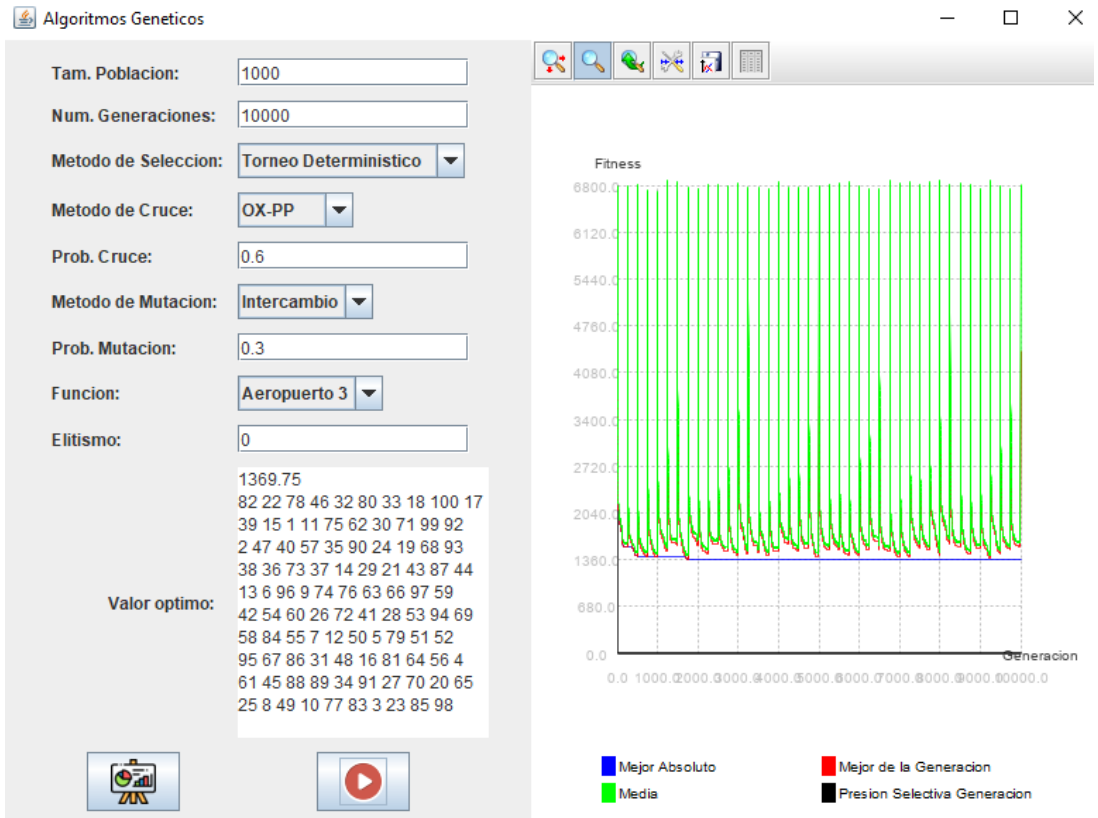
Este aeropuerto tiene más vuelos y pistas, por lo que necesita mayores valores para alcanzar el óptimo. Con métodos malos como estocástico o ruleta necesita un mayor tamaño para alcanzar el óptimo.



Con los mejores métodos que hemos encontrado, obtiene el valor óptimo del aeropuerto 2 rápidamente. Con los Torneos, Truncamiento y Ranking.

4.1.3. Aeropuerto 3





Se reinicia la población cada 250 generaciones, conservando el 10% mejor.

4.2. Análisis

Se pueden ver los ficheros .txt vinculados a cada prueba ejecutada.

Para el **Aeropuerto 1**, todas las funciones dan buenos resultados con pocas generaciones y un tamaño de población pequeño.

- Prueba 1 (sin elitismo): Tamaño de población=50, generaciones=20, probabilidad de cruce y mutación= 0.6 y 0.3.

  Se puede ver en esta prueba que todas las combinaciones posibles de métodos de selección, cruce y mutación dan buenos resultados. Esto se debe a que el problema no tiene muchos datos y llega al óptimo bastante rápido.

- Prueba 2 (con elitismo) mejora mucho y se puede reducir el tamaño de población y las generaciones a 30 y 20.

  Da resultados idénticos a la prueba anterior, ya que elitismo ayuda a mantener a un porcentaje de los mejores y así poder cruzarlos y llegar antes al óptimo.

Para el **aeropuerto 2**, hemos ejecutado el mismo algoritmo para calcular las medias de los resultados, pero esta vez aumentando el tamaño de la población, ya que con un tamaño de población y generaciones bajo no alcanza el óptimo.

- Prueba 1 (sin elitismo): Tamaño de población=100, generaciones=100, probabilidad de cruce y mutación= 0.6 y 0.3.

  Se puede apreciar que los métodos de selección; torneos, truncamiento y ranking da mejores resultados que los demás. Pero el método de mutación heurística da mejores resultados para los peores métodos de selección. También se puede ver que el método de cruce OX da muy malos resultados para todos menos los mejores métodos de selección, que solo lo empeora un poco.

- Prueba 2 (con elitismo): Tamaño de población=500, generaciones=250 y misma probabilidad.

  Mejora los resultados de todas las funciones, al tener un mayor tamaño y mas generaciones. La mejora está en un 21% de media, algunas tienen una mayor mejora, como ruleta y estocástico, y las que daban buenos resultados en la prueba anterior ya encuentran de media el óptimo.

Para el **aeropuerto 3**, hemos ejecutado el mismo algoritmo para calcular las medias de los resultados. Pero esta vez una única vez con un tamaño de población y generaciones de 100. Puesto que es un problema bastante lento. En este enlace están las pruebas ejecutadas. Y aquí hay más capturas de ejecución.

Dan resultados parecidos a los anteriores, es decir, los métodos de selección que mejores medias dan son los de torneo, truncamiento y ranking. Junto con método de cruce OX-PP y mutación por intercambio.

4.3. Conclusiones

Métodos de selección, ordenados de mejor a peor:

- Los dos **torneos**, **truncamiento** y **ranking** son los mejores. Esto es así porque los mejores individuos de la población tienen más probabilidades de ser seleccionados. Por lo que encuentran antes el óptimo.
- **Restos** da resultados parecidos a los métodos anteriores (da resultados un poco peores), funciona bien porque también usa las probabilidades acumuladas.
- **Estocástico universal** y **ruleta**, al ser métodos que dependen más de la aleatoriedad dan peores resultados, pero esto se debe a que necesitan más generaciones para encontrar el óptimo.

Métodos de cruce:

- **OX-PP**, **CX** y **CO**, y **nuestro método personalizado**. Estos métodos cruzan los padres sin tramos, es decir, se intercambian puntos aleatorios, así buscando otras formas de ordenar a los aviones.
- **PMX** y **OX**, cruzan los padres usando tramos, por lo que al importar el orden cambia mucho el individuo.

Los métodos de mutación de **inserción, intercambio** funcionan muy bien para la mayoría de las combinaciones. **Heurística** funciona mejor para los métodos de selección con peores resultados, y para los que mejores resultados da los empeora. El método de **inversión** le pasa lo mismo que con los de cruce, al cambiar por completo un tramo no mejora tanto como los otros. En cambio nuestro **método personalizado** al cambiar un tramo entero de forma aleatoria y buscar otras opciones, si da buenos resultados.

Para concluir la peor combinación que hemos observado es ruleta con OX con inversión. Necesita muchas generaciones para dar buenos resultados.

5. Guía de Uso

Antes de ejecutar el proyecto hay que comprobar si la librería externa, JMathPlot está incluida. En el classpath del proyecto está incluido correctamente, en caso de que no se genere correctamente se puede hacer manual de la siguiente forma: Entrar en la configuración del proyecto, estos pasos son para eclipse:

Acceder a "Properties" del proyecto →"Java Build Path" → "Libraries" → "Add External JARs". E incluir el jmathplot.jar de la carpeta lib del proyecto. El proyecto se ejecuta en la clase Main. Aparece la interfaz y se rellenan los datos.

Los siguientes componentes de JSwing son para configurar la ejecución.

JTextFields:

- Tam. Población y Num. Generaciones: son números naturales. No soporta números negativos.
- Prob. Cruce y Prob. Mutación: son "double", con un intervalo de [0, 1]. Cuanto mayor sea el número mayor es la probabilidad.
- Elitismo es un porcentaje, por lo que es un intervalo de [0, 100]. Cuanto mayor sea el número mayor es el conjunto de elite.

JComboBox:

- Método de Selección, Cruce y Mutación: En estos componentes se eligen los métodos para la ejecución del algoritmo.
- Función: Elegir entre Aeropuerto 1, 2 y 3.

JButton:

- Asignacion: Botón de la izquierda. Se puede presionar al ejecutar el programa. Muestra, para cada pista la asignacion de vuelos, con ID, Nombre, Tiempo de llegada esperada, Tiempo de llegada, retraso, entre el tiempo esperado y el final.
- Run: Botón de la derecha. Se usa para ejecutar el programa con los valores asignados en los componentes mencionados anteriormente.

6. Reparto de Tareas

David ha hecho todos los cruces menos el personalizado, así como las tres primeras mutaciones. Daniel ha solucionado los problemas de ruleta y truncamiento de la práctica anterior, los métodos de cruce y mutación restantes y el JDialog con los resultados de las ejecuciones (asignaciones de vuelos a pistas). Las mejoras las hemos implementado juntos en el laboratorio.