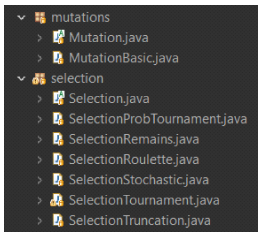
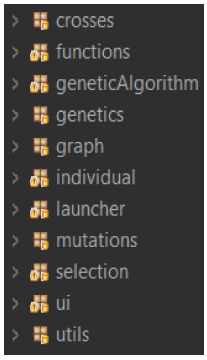
Memoria Práctica 2

- Implementación y estructura



El proyecto está estructurado por paquetes de manera similar a lo explicado en el anexo. El *main* ejecuta la clase '*Interface'* de la práctica, quien se encarga cargar y procesar los valores de entrada a nivel de interfaz, y con ayuda de la clase '*GeneticAlgorithm'* se carga el algoritmo de evolución y se ejecuta el main, devolviendo los valores que se han de insertar en la gráfica.



Respecto a la estructura de las clases, los tipos de selecciones, cruces, mutaciones, genes y funciones se crean partiendo de una clase abstracta de cada tipo de la que heredan, para poder realizar casteos de una manera más cómoda con la interfaz y que no haya problemas de conversión. Además, usamos para cada Función, Selección, Cruce y Mutación un tipo de enum para así hacer más fácil la recolección de datos desde la interfaz. De esta manera creamos una interfaz más genérica para que nos pueda servir de cara a futuras prácticas con solo crear un enum más y una clase más que herede de las abstractas que se menciona anteriormente.

Hablando de los paquetes optamos por tener el código muy troceado, por lo que hay muchos paquetes para dividir las clases de la práctica. Los nombres son intuitivos por lo que no hace falta explicar que hay en cada paquete.

Respecto a la práctica hemos implementado todo lo pedido, tanto los nuevos cruces selecciones y mutaciones hasta el contador de mutaciones

- Guía de Uso

En la primera pestaña se observan unos datos de entrada, aunque también puedes elegir leer los datos de un Case1.txt donde puedes meter la cantidad de aviones que tú quieras (Se encuentra en la carpeta padre donde está el proyecto).

En la segunda pestaña se ven los datos de la gráfica donde puedes elegir el tipo de selección, cruce, mutación, etc como la práctica 1. Además hay nuevas opciones como el fitnessType o el Beta Factor. El primero se usa en la función de adaptación (que se explica más abajo) y el segundo (también conocido como presión selectiva) es un valor que se usa en el método de selección **Ranking**. Además ahora se puede apreciar los valores mínimos y máximos de la población, así como el número de cruces y mutaciones hechos para llegar hasta allí.

En la tercera pestaña se ve la solución de los vuelos, mostrando los vuelos asignados a sus correspondientes pistas y en consola se muestra como están ordenados los vuelos.

La interfaz de la segunda pestaña es igual que la primera. Todas las opciones para modificar se encuentran en la parte izquierda de la ventana. Solo hemos usado Spinners y DropDowns para que sea más cómodo cambiar los valores. Así mismo, el tipo de selección de Truncamiento recibe un parámetro de probabilidad cuando ésta está seleccionada. Además, hemos puesto topes para evitar valores ilógicos al cambiar los parámetros.

- Graficas de Evolución

Las gráficas que vamos a mostrar ahora se han creado con los siguientes parámetros establecidos, variando quizá en alguna probabilidad de cruce o mutación:

Población Inicial	100
Nº Iteraciones	100
% de Cruce	60
% de Mutación	5

El resto de parámetros varía dependiendo de la función, por lo que lo indicaremos al lado de las gráficas. La representación de las gráficas está igual que la pedida en la

práctica, poniendo de color azul el mejor valor absoluto de aptitud, en rojo el mejor valor de aptitud de cada generación, en verde la aptitud media, en naranja la peor aptitud de la población y en purpura el peor absoluto.

FUNCION Practica 2 Aviones

La función de fitness que hemos implementado se basa en la estructura que se muestra en el libro de ***Algoritmos Evolutivos: un enfoque práctico***. Tiene algunos cambios ya que, al ser pseudocódigo, la estructura cambia.

```
for(int i = 0; i < individuo.size(); i++) {
    int pista_asignada = 0;

    FlightGen vuelo = (FlightGen) individuo.get(i);

    //Tipo del avion que ya está en la pista (anterior)
    //Para cada pista del vuelo i-ésimo. Se calcula el menor TLA
    List<Double> tel_pistas_cpy = info_vuelos_.get(vuelo.pos_vuelo).TTEL_vuelo;

    //Menor tla de las 3 pistas
    double menor_tla = Double.POSITIVE_INFINITY;
    //TEL definitivo a la pista asignada
    double TEL_vuelo = Double.POSITIVE_INFINITY;
    FlightType typeCurrent_ = info_vuelos_.get(vuelo.pos_vuelo).type_;
    for(int j = 0; j < tel_pistas_cpy.size(); j++) {
        //Tipo del avion que viene
        //TEL de la pista j-ésima
        double TEL_pista = tel_pistas_cpy.get(j);
        //Calculo del menor TEL de las 3 pistas
        if(menor_tel_type && TEL_pista < TEL_vuelo)  TEL_vuelo = TEL_pista;

        //Var aux
        double TLA = 0;
        if(TLA_Pistas.get(j).size() == 0) { // es el primer vuelo, no hay vuelos en esa pista
                        //tla 0-> no hay tla anterior porque no hay ningun avion en la pista
                        //sep 0-> al no haber avion, no hay separacion
            TLA = Math.max(0 + 0, TEL_pista);
        }else{
            FlightType typeBefore_ = TLA_Pistas.get(j).get(TLA_Pistas.get(j).size() - 1).type;
            //Separacion del vuelo actual con el que ya esta en pista (actual con anterior)
                                        //Pista j-ésima | ultimo valor/vuelo añadido a esa pista
            double sep = separations_.get(typeBefore_.ordinal()).get(typeCurrent_.ordinal());
            double beforeTLA = TLA_Pistas.get(j).get(TLA_Pistas.get(j).size() - 1).TLA;
            //maximo del TLA anterior mas la separacion | Tel de esa pista
            TLA = Math.max( beforeTLA + sep, TEL_pista);
        }

        //Nos quedamos con el menor TLA (minimización)
        if(TLA < menor_tla) {
            menor_tla = TLA;
            pista_asignada = j;
        }
    }

    //Añadimos el vuelo j-esimo a la pista asiganda (con menor TLA)
    TLA_Pistas.get(pista_asignada).add(new TAsignacion(menor_tla, typeCurrent_));
    vuelo.pistaAsignada = pista_asignada;
    vuelo.TLA = menor_tla;
    //Calculo del TEL que es basicamente su TEL de la pista asignada
    if(!menor_tel_type) TEL_vuelo = tel_pistas_cpy.get(pista_asignada);

    //Suma de fitness
    fitness += Math.pow(vuelo.TLA - TEL_vuelo, 2);
}
```

Básicamente por cada vuelo del individuo vamos a calcular qué pista nos proporciona un menor Tiempo de Llegada Asignado, respetando siempre las restricciones de separación por pesos. Para ello realizamos una comparación por cada pista con los aviones (anteriores) que ya han llegado a la pista y nos quedamos con la menor. Algo tal que así: max (separación + TLA Anterior, Tiempo de Llegada Estimado).

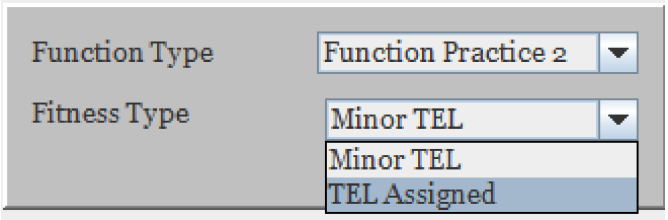
Al quedarnos con el menor, también nos quedamos con la pista que ha dado menor tiempo, para luego introducir el vuelo en dicha pista. Asignamos el TLA y Pista al vuelo y posteriormente calculamos parte del fitness.

La adaptación se calcula de la siguiente manera (TLA - TEL)^2. Este cálculo se realiza por cada individuo y se va acumulando, para luego ser devuelto. Cabe recalcar que el TEL puede tomar dos valores:

- Puede ser el menor TEL de cualquiera de las tres pistas.

- Puede ser el TEL de la pista al cual ha sido asignado.

Esto se puede elegir en la interfaz, donde hay un desplegable tal que así:



- **Conclusiones**
  - **Métodos de Selección**

    Hemos conseguido que todos los métodos de selección den el valor exacto de aptitud, ha excepción del PMX, del cuál más adelante hablaremos de él. Por lo demás ninguna destaca.

  - **Cruce**

    Con los cruces nos ha pasado lo mismo, todos los cruces obtienen el valor exacto de la aptitud.

  - **Elitismo**

    No hay una diferencia notoria entre el elitismo o sin él, lo que sí que se observa es que gracias al elitismo hay una clara mejoría exponencial, y se llega al valor de aptitud con muchas menos generaciones.
- **Reparto de tareas**

Tras tener la práctica 1 como base para comenzar esta no hemos tenido mucha dificultad a la hora de realizar la estructura de la práctica. Hemos seguido trabajando por GitHub con un ritmo constante, dividiéndonos el trabajo correctamente, aunque el integrante José Daniel ha trabajado más ya que él solo consiguió la función de adaptación, siendo esto lo más complejo de la práctica, pues el resto era implementar los cruces y mutaciones vistos en clase. Samuel Blázquez se ha encargado de implementar los cruces y mutaciones. Sin embargo, no hemos conseguido que el cruce PMX realice su trabajo bien, pues a pesar de que a simple vista (y en las primeras generaciones) realice el cruce correctamente, posteriormente se aprecia que hay genes repetidos y/o salten excepciones.

- **Información Extra**

Queremos recalcar que si bien el caso base del enunciado nos proporciona un fitness óptimo (**11.25**), el orden de los aviones no es el que se ve en el enunciado, sino que uno distinto. Entendemos que puede haber mas de una solución. Los valores de la tabla si son correctos. Al añadir otros dos casos de prueba, se aprecia que conforme el número de aviones va creciendo, el valor de fitness varia más, es decir, hay más ruido y no se mantiene en un valor estable. Por ejemplo, para 14 vuelos, el fitness es de 14.6-15, pero para 20 vuelos, los valores varían desde 80 hasta 140.

Otro detalle al recalcar es que por errores de la Practica 1 (fallos de concepto en algunos métodos de selección que fueron resueltos en la reentrega de ésta), nos ha consumido tiempo para el desarrollo de la Práctica 2. Ya nos sabemos manejar mucho mejor en el lenguaje Java y Eclipse pero los errores anteriores nos han perjudicado considerablemente, siendo así ésta última semana en la que realmente hemos trabajo completamente en esta Práctica y nos ha faltado algo de tiempo, pues aparte de esto tenemos entregas de prácticas y exámenes hasta el jueves 7, por lo que habíamos pensado en acabar de corregir la práctica y todos los fallos una vez tuviéramos tiempo para mirarlo más detenidamente.

ACTUALIZACIÓN:

Hemos solucionado los errores de PMX, ya realiza el cruce correctamente sin repetición de vuelos ni excepciones. También se ha corregido el cruce codificación ordinal que provocaba en ocasiones aviones repetidos. Por los demás, todo sigue igual.