



UNIVERSIDAD DE BURGOS

COMPUTACIÓN NEURONAL Y EVOLUTIVA

P4: AirTrafficController

Analizar y desarrollar un algoritmo genético que realice el cálculo automático de la distribución más conveniente de vuelos que solicitan aterrizar en un determinado aeropuerto.

Estudiantes:

DAVID MIGUEL LOZANO
JAVIER MARTÍNEZ RIBERAS

Profesor de la asignatura:

BRUNO BARUQUE ZANÓN

1º semestre 2016

Índice

A. Introducción	2
B. Solución propuesta	2
B.1. Representación de los individuos	2
B.2. Esquema evolutivo	3
B.3. Función de fitness	3
B.4. Inicialización	3
B.5. Criterio de parada	4
B.6. Criterio de selección	4
B.7. Operador de cruce	4
B.8. Operador de mutación	4
B.9. Criterio de reemplazo	4
B.10. Implementación	5
C. Análisis	5
C.1. Resultados con los diferentes ficheros de prueba	5
C.2. Resultados con los diferentes operadores genéticos	6
C.2.1. Selección: RouletteSelector vs. RandomSelector	6
C.2.2. Cruce: OrderOXCrossover vs. OrderPMXCrossover	7
C.2.3. Mutación: Order2OptMutator vs. OrderSublistMutator	8
C.3. Resultados con variación de los parámetros	8
C.3.1. Tamaño de la población	9
C.3.2. Probabilidad de cruce	9
C.3.3. Probabilidad de mutación	10
C.4. Análisis	10
D. Conclusiones	10

A. Introducción

El objetivo de esta práctica es desarrollar un programa que implemente un algoritmo genético que permita al usuario realizar el calculo automático de la distribución más conveniente de vuelos que solicitan aterrizar en un determinado aeropuerto. Así mismo, se analizará la conveniencia de la solución propuesta y se reflexionará sobre los resultados obtenidos.

La aplicación tiene que ser capaz de adaptarse a las siguientes condiciones:

- El número de pistas del aeropuerto debe ser un parámetro configurable.
- El número de aviones en cada situación puede variar.
- Los aviones tienen un programa de vuelo que incluye:
 - ETA (*Estimated Time of Arrival*): tiempo estimado de llegada calculado en el momento del despegue.
 - Tipo de avión: heavy / big / small. Condiciona el tiempo necesario para su aterrizaje.

El programa tiene que conseguir obtener de forma automática la mejor asignación posible de vuelos a aterrizar en cada pista, de forma que el tiempo de espera de los vuelos en su conjunto sea el menor posible.

Para su resolución se hará uso de la librería para Java JCLEC (Java Class Library for Evolutionary Computation) [5]. La cual, proporciona un framework para programación evolutiva que da soporte, entre otras cosas, a los algoritmos genéticos.

B. Solución propuesta

A continuación se detalla la codificación y configuración del problema para ser resuelto con un algoritmo genético.

B.1. Representación de los individuos

Para representar los individuos se ha utilizado un array de enteros ordenado (`OrderArrayIndividual`). Se probaron dos representaciones diferentes, en ambas cada posición del array representaba un avión, pero el ordenamiento era distinto:

1. Array ordenado por orden de llegada. De tal manera, que la primera posición se correspondía con el primer avión en llegar. Y el valor de cada posición indicaba el número del avión.

2. Array ordenado por número de avión. De tal manera, que la primera posición se correspondía con el avión número uno. Y el valor de cada posición indicaba la posición de llegada del avión.

Tras realizar pruebas, se vió que los resultados eran muy similares. Por lo que se eligió la representación 1 para realizar el estudio.

Ejemplo de genotipo:

$$\begin{bmatrix} 2 & 3 & 1 & 4 \end{bmatrix}$$

Representa que el primer vuelo en aterrizar fue el 2, seguido del 3, 1 y 4.

B.2. Esquema evolutivo

Se ha utilizado el algoritmo SGE (*Simple Generational and Elitist*). Se trata de un algoritmo elitista que asegura que, en cualquier momento, sólo los mejores individuos pasen a la siguiente generación [4].

B.3. Función de fitness

Para evaluar los individuos, como el genotipo se encontraba ordenado por orden de llegada, se iba iterando sobre él y planificando cada vuelo. La asignación de la pista se realizaba minimizando el ATA, de tal forma, que se asignaba la primera pista que quedase libre. Por último, el cálculo del fitness se realizó de dos maneras:

1. Minimizando el retraso acumulado. Es decir, el sumatorio de la diferencia entre el ATA y el mínimo ETA de cada avión.
2. Minimizando el instante de llegada del último aterrizaje.

Se compararon ambos métodos y se vió que arrojaban resultados similares. Sin embargo, el método 2 tenía una varianza mucho más grande que el 1. Por este motivo, se eligió el método 1 para el estudio.

B.4. Inicialización

La población inicial se inicializa de forma aleatoria. Se ha utilizado el generador **Ranecu**, se trata de un generador lineal congruencial avanzado con un periodo aproximado de 10^{18} [2].

B.5. Criterio de parada

El criterio de parada se ha establecido en 1.000 generaciones.

B.6. Criterio de selección

Para seleccionar un subconjunto de la población se han analizado los siguientes algoritmos:

1. `RouletteSelector`: selección por ruleta [3].
2. `RandomSelector`: selección aleatoria [1].

B.7. Operador de cruce

Para obtener un nuevo individuo basado en el genotipo de sus padres se han analizado los siguientes algoritmos:

1. `OrderOXCrossover`: OX Crossover.
2. `OrderPMXCrossover`: PMX Crossover.

La probabilidad de cruce se estableció en un 75 %.

B.8. Operador de mutación

Cada gen del genotipo de un individuo tiene 3 % de probabilidad de mutar. Se han analizado los siguientes algoritmos de mutación:

1. `Order2OptMutator`: mutación 2-opt del genotipo.
2. `OrderSublistMutator`: mutación de una sublista del genotipo aleatoriamente.

B.9. Criterio de reemplazo

Se ha utilizado `OrderArrayCreator`, mediante el cual, los hijos reemplazan directamente a los padres. Para preservar el elitismo, si la mejor solución de la generación anterior no sobrevive, la peor solución se reemplaza por una nueva.

B.10. Implementación

Para la importación de la librería JCLEC se ha creado una dependencia Maven de esta. Se ha publicado en el siguiente repositorio: [JCLEC Maven Repository](#).

La aplicación cuenta con las siguientes clases:

- **Run**: permite lanzar la aplicación seleccionando por parámetro el archivo de vuelos deseado.
- **AirTrafficController**: implementación del algoritmo genético.
- **Airport**: clase que modela un aeropuerto. Posee la lógica para seleccionar la mejor pista para un determinado avión. Además, permite conocer el retraso acumulado o el momento en el que aterrizó el último avión.
- **Runway**: clase que modela una pista del aeropuerto. Posee la lógica para calcular cuando estará libre para que aterrice un determinado tipo de avión.
- **Flight**: clase que modela un vuelo. Posee la lógica para calcular el retraso que tuvo.

C. Análisis

A continuación exponemos los resultados obtenidos y realizamos un análisis de estos.

C.1. Resultados con los diferentes ficheros de prueba

Se ejecutó el algoritmo con los diferentes ficheros de prueba y la siguiente configuración fija:

- Selección: `RouletteSelector`.
- Cruce: `OrderPMXCrossover`.
- Mutación: `Order2OptMutator`.

En la siguiente tabla se muestran los fitness obtenidos para cada uno de los test junto con el instante en el que se realizó el último aterrizaje:

Fichero	Mejor	Peor	Medio	Último aterrizaje
IncomingFlights_1	30	119	114	13
IncomingFlights_2	215	689	595	47
IncomingFlights_3	27	172	170	19
IncomingFlights_4	147	531	382	16

Cuadro 1: Resultados ficheros de test

Todos los gráficos y logs generados se encuentran disponibles en: TODO

C.2. Resultados con los diferentes operadores genéticos

A continuación se exponen los resultados de comparar diferentes implementaciones de los operadores genéticos. El archivo de pruebas utilizado fue `IncomingFlights_4`.

C.2.1. Selección: `RouletteSelector` vs. `RandomSelector`

El resto de parámetros se fija a:

- Cruce: `OrderPMXCrossover`.
- Mutación: `Order20ptMutator`.

En la siguiente tabla se muestran los fitness obtenidos para cada uno de los test junto con el instante en el que se realizó el último aterrizaje:

Algoritmo	Mejor	Peor	Medio	Último aterrizaje
<code>RouletteSelector</code>	147	531	328	16
<code>RandomSelector</code>	90	427	259	14

Cuadro 2: `RouletteSelector` vs. `RandomSelector`

Los gráficos de la ejecución fueron:

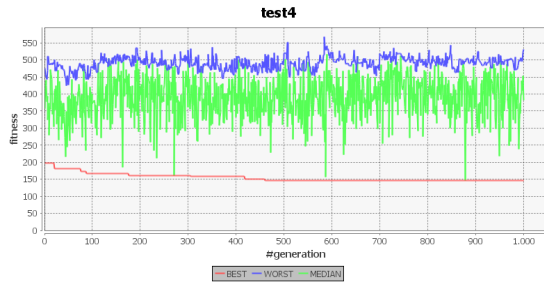


Figura 1: RouletteSelector

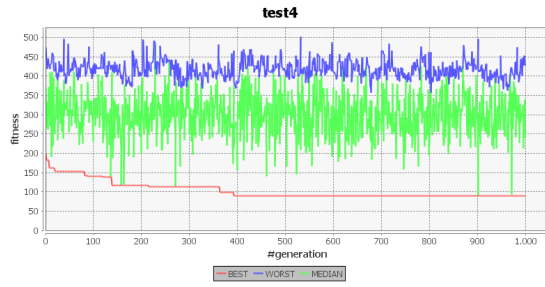


Figura 2: RandomSelector

C.2.2. Cruce: OrderOXCrossover vs. OrderPMXCrossover

El resto de parámetros se fija a:

- Selección: RouletteSelector.
- Mutación: Order20ptMutator.

En la siguiente tabla se muestran los fitness obtenidos para cada uno de los test junto con el instante en el que se realizó el último aterrizaje:

Algoritmo	Mejor	Peor	Medio	Último aterrizaje
OrderOXCrossover	137	487	290	17
OrderPMXCrossover	147	531	382	16

Cuadro 3: OrderOXCrossover vs. OrderPMXCrossover

Los gráficos de la ejecución fueron:

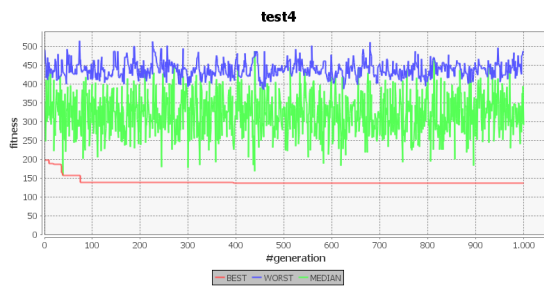


Figura 3: OrderOXCrossover

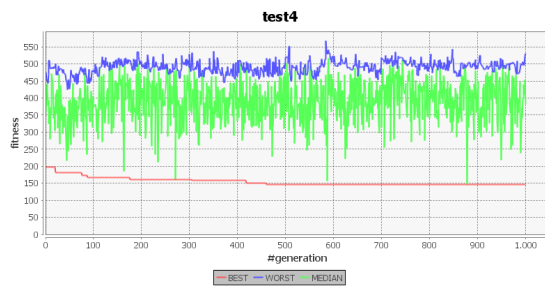


Figura 4: OrderPMXCrossover

C.2.3. Mutación: Order2OptMutator vs. OrderSublistMutator

El resto de parámetros se fija a:

- Selección: `RouletteSelector`.
- Cruce: `OrderPMXCrossover`.

En la siguiente tabla se muestran los fitness obtenidos para cada uno de los test junto con el instante en el que se realizó el último aterrizaje:

Algoritmo	Mejor	Peor	Medio	Último aterrizaje
Order2OptMutator	147	531	382	16
OrderSublistMutator	150	487	388	17

Cuadro 4: Order2OptMutator vs. OrderSublistMutator

Los gráficos de la ejecución fueron:



Figura 5: Order2OptMutator

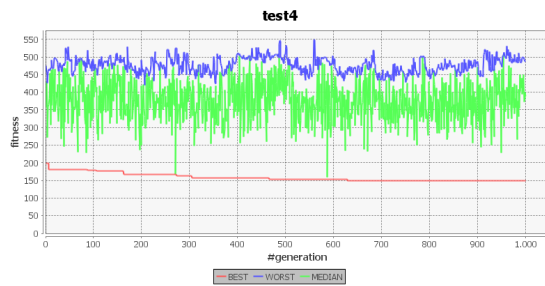


Figura 6: OrderSublistMutator

C.3. Resultados con variación de los parámetros

A continuación se exponen los resultados de variar ciertos parámetros numéricos del algoritmo. El archivo de pruebas utilizado fue `IncomingFlights_4`. Las implementaciones de los operadores genéticos fueron:

- Selección: `RouletteSelector`.
- Cruce: `OrderPMXCrossover`.
- Mutación: `Order2OptMutator`.

C.3.1. Tamaño de la población

Tamaño	Mejor	Peor	Medio	Último aterrizaje
50	158	465	327	16
500	112	557	508	15

Cuadro 5: 50 vs. 500



Figura 7: 50



Figura 8: 500

C.3.2. Probabilidad de cruce

Probabilidad	Mejor	Peor	Medio	Último aterrizaje
50 %	135	529	440	15
90 %	128	429	389	17

Cuadro 6: 50 % vs. 90 %

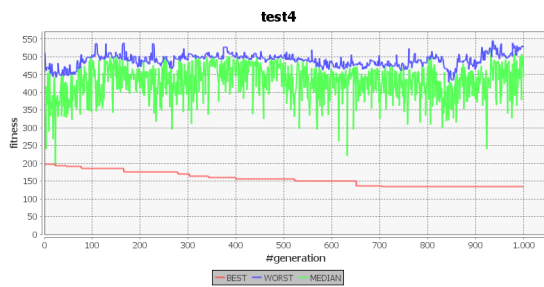


Figura 9: 50 %

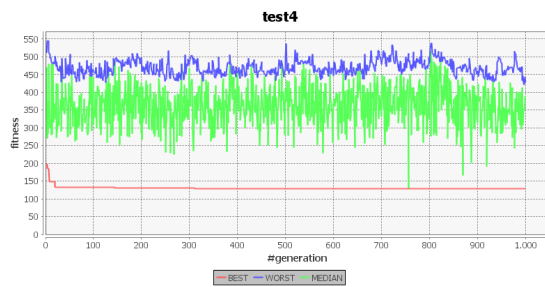


Figura 10: 90 %

C.3.3. Probabilidad de mutación

Probabilidad	Mejor	Peor	Medio	Último aterrizaje
5 %	118	473	409	18
20 %	126	488	308	17

Cuadro 7: 5 % vs. 20 %



Figura 11: 5 %



Figura 12: 20 %

C.4. Análisis

TODO

D. Conclusiones

TODO

Referencias

- [1] JCLEC. Class randomselector, 2008. URL <http://jclec.sourceforge.net/data/jclec4-classification-doc/net/sf/jclec/selector/RandomSelector.html>.
- [2] JCLEC. Class ranecu, 2008. URL <http://jclec.sourceforge.net/data/jclec4-classification-doc/net/sf/jclec/util/random/Ranecu.html>.
- [3] JCLEC. Class roulettselector, 2008. URL <http://jclec.sourceforge.net/data/jclec4-classification-doc/net/sf/jclec/selector/RouletteSelector.html>.
- [4] JCLEC. Class sge, 2008. URL <http://jclec.sourceforge.net/data/jclec4-classification-doc/net/sf/jclec/algorithm/classic/SGE.html>.
- [5] Sebastián Ventura, Cristóbal Romero, Amelia Zafra, Jose Antonio Delgado, and César Hervás. JCLEC - java class library for evolutionary computation, 2008. URL <http://jclec.sourceforge.net/>.