

Algoritmos Genéticos (Opt4J)

Nombre: _Elias Urios Alacreu

Tiempo: 1 hora.

Importante: subid a Poliformat el código generado para cada pregunta (por ejemplo en un .zip). Se utilizará como base el ejercicio multi-objetivo propuesto en la práctica.

1. (2 puntos) Debido a una modificación en la normativa y control de las potencias, se permite que se exceda la potencia máxima contratada hasta en un máximo del 10% en cualquiera de los tramos (puede ser en uno, dos o los tres). Por ejemplo, ahora será posible que en el tramo 1 se alcancen los $15+1.5 = 16.5$ kW. Si se requiere esa potencia extra habrá que pagar una penalización de 5€ (a restar del beneficio global), independientemente del número de tramos en los que se haya utilizado esa potencia.

Explica el cambio realizado e indica cuál es el conjunto de mejores soluciones encontradas para este nuevo problema tras la ejecución de 800 iteraciones. El resto de los parámetros se deja a libertad del alumno (**NOTA:** indicad claramente cuáles son estos parámetros).

En el evaluador:

```
public Objectives evaluate(ArrayList<Integer> fenotipo) {
    ...
    boolean potencia_extra = false; //indica si usamos potencia extra del 10

    for(int i = 0; i < fenotipo.size(); i++) {
        tramo = fenotipo.get(i);
        turnos[tramo]++;
        if (tramo > 0) { // tramo 0 es ficticio, no nos sirve de nada
            potencia = calcular_potencia(i); //sacamos la potencia que consume dicho
producto
            current_potencia[tramo] += calcular_potencia(i); //potencia consumida hasta
ahora

            if(!potencia_OK(current_potencia[tramo],
DatosElectricidad.potenciaMaximaTramo[tramo]*1.1)) { // potencia_actual > potencia_maxima?
                beneficio_total = Integer.MIN_VALUE; //no es posible esta situacion, el
beneficio es -inf
                turnos[tramo] = Integer.MIN_VALUE; // apartado 2), si no es buena
solucion no hemos fabricado
                break;
            }

            if (current_potencia[tramo] > DatosElectricidad.potenciaMaximaTramo[tramo])
                potencia_extra = true;

            beneficio_total += calcular_beneficio(i, tramo, potencia); //incrementamos el
beneficio total
        }
    }
    //tramos de la mañana
    int punta = 1;
```

```

        int llano = 2;

        if(potencia_extra)
            beneficio_total -= 5;

        Objectives objectives = new Objectives();
        objectives.add("Beneficio total obtenido", Sign.MAX, beneficio_total);
        objectives.add("Numero de productos fabricados en los turnos diurnos", Sign.MAX, turnos[punta] +
turnos[llano]);
        return objectives;
    }

```

Parametros:
 Generations: 800
 populationSize: 100
 parentsPerGeneration:25
 offspringsPerGeneration:25
 crossoverRate: 0.95

Conjunto de mejores soluciones:

#	Individual	Beneficio	Uds.
1	[1, 1, 2, 1, 2, 3, 2, 1, 3, 2, 3, 3, 2, 1, 0, 1, 2, 3, 3, 1]	51.272	13.0
2	[2, 2, 1, 1, 2, 1, 3, 2, 1, 2, 3, 3, 2, 1, 2, 1, 3, 3, 3, 2]	48.248	14.0

Realiza varias pruebas del algoritmo genético modificando los parámetros “tamaño de la población” y “número de iteraciones”. De acuerdo a las pruebas que has realizado, ¿resulta más adecuado trabajar con una población de mayor tamaño o realizar más iteraciones? Razona la respuesta en base a tus experimentos.

Conforme vamos aumentando el tamaño de población, pero manteniendo el numero de iteraciones constantes el resultado se va alejando del mejor valor que hemos observado en los experimentos (beneficio 51.272 y 13 uds). Por lo que conviene incrementar el numero de iteraciones puesto que a partir de un determinado numero (para mis ejecuciones, 1500) siempre produce este resultado que es uno de los mejores.

2. (3.5 puntos) **A partir de las modificaciones del ejercicio 1**, se ha detectado que ciertas combinaciones de fabricación de productos en el mismo tramo afectan al beneficio global, pues es necesario/innecesario hacer ajustes en la maquinaria. Concretamente:
- Si en el mismo tramo (1, 2 o 3) se fabrican P1, P3 y P5, el beneficio se reduce en 4€.
 - Si en el mismo tramo (1, 2 o 3) se fabrican P2 y P4, el beneficio aumenta en 3€.

Explica el cambio realizado e indica cuál es el conjunto de mejores soluciones que encuentras para este nuevo problema tras la ejecución de 800 iteraciones. El resto de los parámetros se deja a libertad del alumno (**NOTA:** indicad claramente cuáles son estos parámetros).

NOTA DE IMPLEMENTACIÓN. Recordad que las colecciones en Java comienzan con el índice 0.

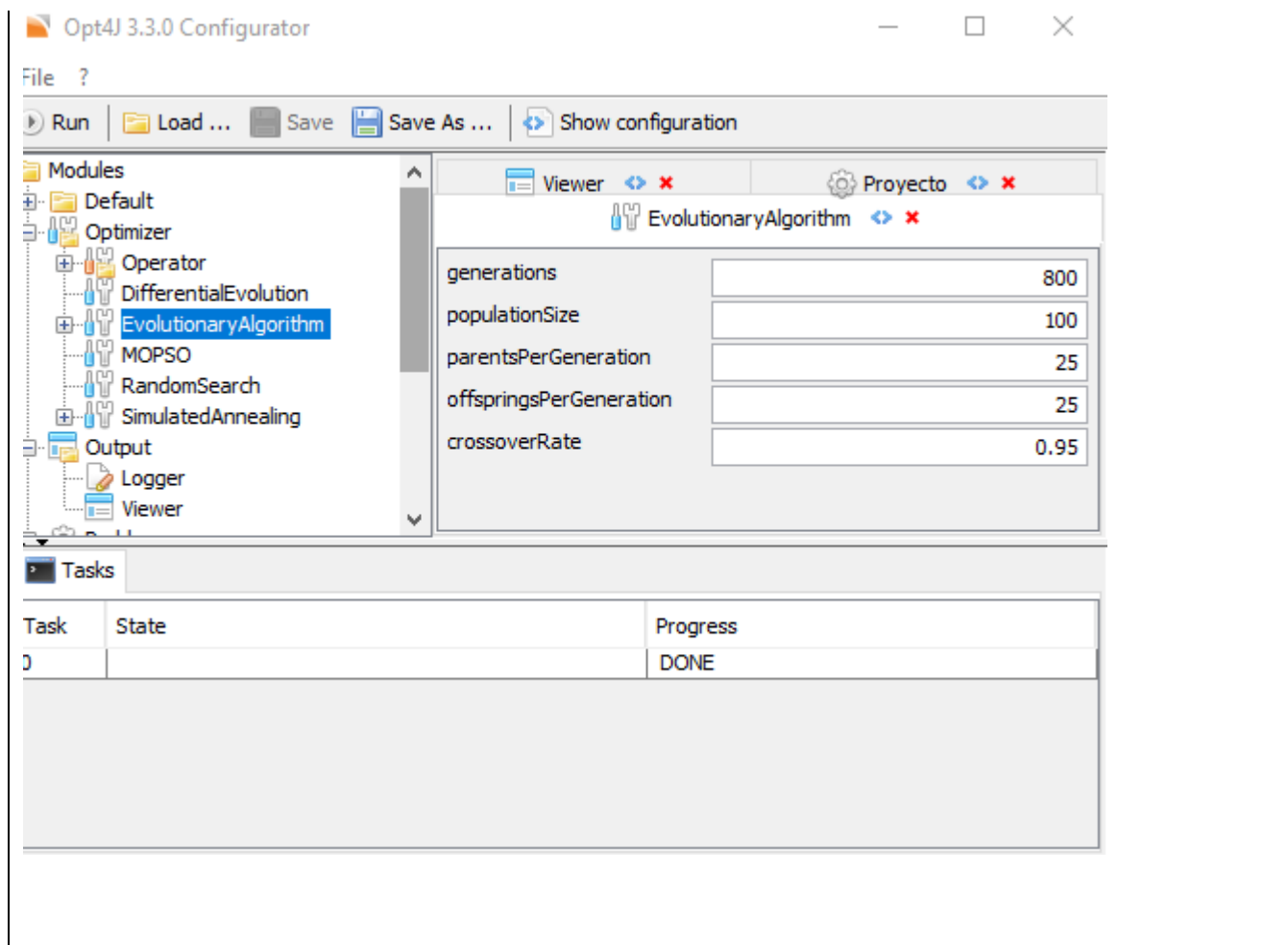
Hemos añadido 4 variables nuevas, dos variables contador que nos indican si los productos mencionados se fabrican todos en el mismo tramo y luego dos variables tramo*i* donde *i*={1,2} que indican donde se fabrican, respectivamente, P1 y P2. Luego solo hace faltar comprobar que se cumplen las condiciones

En el evaluador:

```
public Objectives evaluate(ArrayList<Integer> fenotipo) {
    ...
    int contador1 = 0;
    int contador2 = 0;
    int tramo1 = -1;
    int tramo2 = -1;
    for(int i = 0; i < fenotipo.size(); i++) {
        ....
        tramo = fenotipo.get(i);
        if(i==0)
            tramo1=tramo;
        if(i==1)
            tramo2=tramo;
        if(tramo > 0) {
            ...
            if(tramo1 > 0 && (i == 0 || i == 2 || i == 4) && tramo1 == tramo)
                contador1++;
            if(tramo2 > 0 && (i == 1 || i == 3) && tramo2 == tramo)
                contador2++;
        }
    }
    ...
    if(contador1 == 3)
        beneficio_total -= 4;
    if(contador2 == 2)
        beneficio_total += 3;
    ...
    //PD: Donde hay “...” es el código original
}
```

#	Individual	beneficio uds.	
1	[2, 2, 3, 2, 2, 2, 1, 1, 2, 1, 3, 3, 2, 1, 0, 1, 2, 3, 3, 3]	54.084	13.0
2	[2, 2, 3, 2, 1, 2, 3, 2, 2, 1, 3, 3, 1, 1, 2, 1, 2, 3, 3, 1]	51.04	14.0

Parametros:



3. (3.5 puntos) **A partir de las modificaciones del ejercicio 2**, deseamos añadir un nuevo Tramo4 que modifica el Tramo3. La información de los Tramos 1 y 2 se mantiene igual, y los cambios son:

	Tramo3	Tramo4
Nombre	Super-reducido	Reducido
Horario	06:00-08:00	08:00-10:00
Potencia máxima contratada (kW)	27	25
Precio del kWh (€)	0.1	0.15

Además, deseamos poder fabricar 2 productos más (P21 y P22) con la siguiente información:

	P21	P22
Consumo (kWh)	3.3	4.2
Beneficio (€)	5	7

Explica los cambios realizados y cuál es el conjunto de mejores soluciones para este nuevo problema. Todos los parámetros se dejan a libertad del alumno (**NOTA:** indicad claramente cuáles son estos parámetros).

Simplemente tenemos que modificar el fichero DatosElectricidad.java donde se encuentran todos los datos del problema, hay que cambiar las diversas variables que hay (NUM_TRAMOS = 4, añadir una entrada nueva a horasTramos, hasta enumerar todas las características que nos piden). Y nos queda algo así:

```
package proyecto;
```

```
public class DatosElectricidad
```

```
{

    public static final int NUM_TRAMOS      = 4;

    public static final int NUM_PEDIDOS     = 22;

    // consumo para los 20 pedidos

    public static final double[] consumo =

        {

            1.9, 2.1, 3.0, 0.7, 1.5, 3.3, 4.2, 2.6, 2.3, 3.2, 4.5, 4.2, 2.7, 1.9, 3.5, 2.7, 3.4, 4.5, 6.2, 2.3,
3.3, 4.2

        };

    // beneficio para los 20 pedidos

    public static final double[] beneficio =

        {

            3, 5, 6, 1, 3, 4, 9, 3, 4, 4, 8, 7, 4, 3, 4, 5, 4, 6, 9, 4, 5, 7

        };

    // horas de cada uno de los tramos (el tramo 0 es ficticio)

    public static final int[] horasTramo =

        {

            0, 4, 4, 2, 2

        };

}
```

```
// potencia maxima contratada para cada tramo (el tramo 0 es ficticio)
```

```
public static final double[] potenciaMaximaTramo =
```

```
{
```

```
0, 15, 18, 27, 25
```

```
};
```

```
// precio del kWh para cada tramo (el tramo 0 es ficticio)
```

```
public static final double[] preciokWhTramo =
```

```
{
```

```
0, 0.26, 0.18, 0.1, 0.15
```

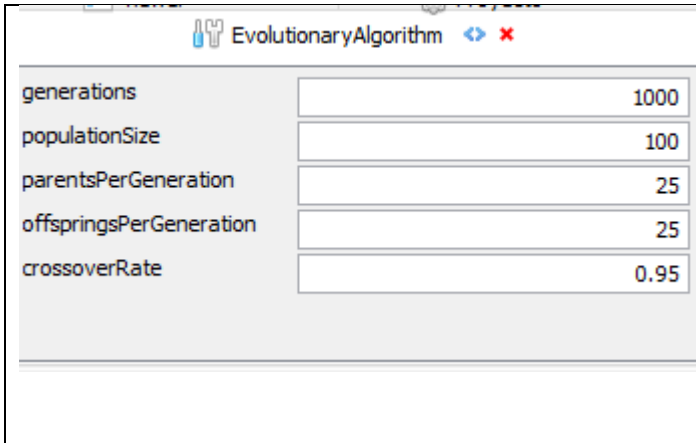
```
};
```

```
}
```

Los mejores resultados obtenidos son (mismas columnas como en los anteriores apartados):

#	Individual	Beneficio	Uds
1	[2, 2, 4, 2, 4, 4, 4, 2, 2, 3, 3, 3, 2, 2, 4, 4, 4, 3, 3, 2, 4, 3]	86.28999999999999	8.0
2	[2, 2, 4, 2, 2, 4, 4, 2, 2, 3, 3, 3, 2, 2, 4, 4, 4, 3, 3, 2, 4, 3]	85.65999999999998	9.0
3	[2, 2, 4, 2, 2, 4, 4, 2, 2, 3, 3, 3, 2, 2, 4, 1, 4, 3, 3, 2, 4, 3]	83.66199999999999	10.0
4	[1, 2, 2, 2, 1, 4, 4, 2, 2, 3, 3, 4, 2, 2, 4, 2, 3, 3, 3, 1, 4, 3]	81.36200000000001	11.0
5	[1, 2, 2, 2, 2, 4, 4, 1, 1, 2, 3, 3, 2, 2, 4, 2, 3, 3, 3, 1, 4, 3]	79.03	12.0
6	[1, 2, 1, 2, 2, 2, 4, 2, 1, 2, 3, 3, 2, 2, 4, 1, 3, 3, 3, 1, 4, 3]	76.652	13.0
7	[1, 2, 2, 2, 1, 1, 4, 2, 1, 2, 3, 3, 2, 2, 4, 1, 3, 3, 3, 1, 2, 3]	69.68999999999998	14.0

Los parámetros usados:



Parameter	Value
generations	1000
populationSize	100
parentsPerGeneration	25
offspringsPerGeneration	25
crossoverRate	0.95

4. (1 punto) Explica razonadamente (**no es necesario implementar nada**) cuál sería el mejor genotipo si simplemente se deseara obtener una solución que indicara si un producto se va a fabricar en un tramo diurno o nocturno. Es decir, nos da igual saber qué turno concreto se está usando y simplemente nos bastaría saber si es turno diurno o no.

El mejor genotipo sería **BooleanGenotype**, puesto que nos permite expresar una representación binaria de forma sencilla donde si un producto se fabrica de día se puede expresar como que su valor es **true** y, en el caso de que se fabrique de noche se puede expresar como **false** (o del revés, como desee implementarlo cada programador). La idea clave es que como solo hay dos posibilidades, lo podemos representar ante una secuencia de valores T/F.