Técnicas, Entornos y Aplicaciones de Inteligencia Artificial

Algoritmos Genéticos (Opt4J)

Nombre: David Arnal García

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.
Duración: 1 hora

1. (2 puntos) Debido a una modificación en la normativa, que pretende incentivar una reducción en el consumo, si la potencia utilizada en cualquiera de los tramos (puede ser en uno, dos o los tres) se queda por debajo del 90% de la potencia máxima contratada se realizará una única bonificación de 10€. Por ejemplo, si en el tramo 1 se alcanza un consumo por debajo de 0.9*15 = 13.5 kW, se obtendrá dicha bonificación (a sumar al beneficio global), independientemente de si se ha conseguido en uno o más tramos.

   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).

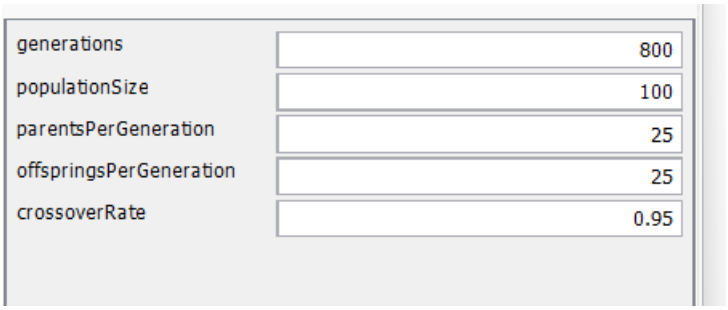
En el *evaluator*:

```
public class Ejercicio1 implements Evaluator<ArrayList<Integer>> {
	public Objectives evaluate(ArrayList<Integer> phenotype) {
		double beneficio = 0;
		int productos = 0;
		double[] potenciaActual = new double[DatosElectricidad.NUM_TRAMOS + 1];
		int tramo;
		int bono = 10;
		boolean existeBono = false;
		for (int i = 0; i < phenotype.size(); ++i) {
			tramo = phenotype.get(i);
			// Condicion tener suficiente potencia en ese tramo
			if (tramo != 0 && potenciaActual[tramo]
					+ DatosElectricidad.consumo[i] >
DatosElectricidad.potenciaMaximaTramo[tramo]) {
				// Si no hay suficiente potencia se deja para la siguiente
jornada laboral
				// penalizando su beneficio
				phenotype.set(i, 0);
				tramo = 0;
			}

			// Para aquellos productos que no se han dejado para la proxima
jornada laborar
			if (tramo > 0) {
				potenciaActual[tramo] += DatosElectricidad.consumo[i];
				if (tramo == 1 || tramo == 2) {
					++productos;
				}
				beneficio += DatosElectricidad.beneficio[i] -
(DatosElectricidad.consumo[i]
						* DatosElectricidad.horasTramo[tramo] *
DatosElectricidad.preciokWhTramo[tramo]);
			}
```

```
            }
            for (int i = 1; i < DatosElectricidad.NUM_TRAMOS; ++i) {
                  tramo = phenotype.get(i);
                  if (potenciaActual[tramo] <
DatosElectricidad.potenciaMaximaTramo[i] * 0.9) {
                        existeBono = true;
                  }
            }

            if (existeBono) {
                  beneficio += bono;
            }

            Objectives objectives = new Objectives();
            objectives.add("Beneficio-MAX", Sign.MAX, beneficio);
            objectives.add("Productos-MAX", Sign.MAX, productos);
            return objectives;
      }

}
```

Individuo Beneficio Unidades

1 [2, 1, 1, 2, 1, 2, 3, 2, 1, 3, 3, 3, 3, 1, 0, 2, 1, 2, 3, 2] 61.272 13



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.

Según se va aumentando el tamaño de la población con el mismo número de iteraciones, el resultado cada vez va empeorando. Por tanto, la mejor opción es aumentar el número de iteraciones, ya que, a partir de un determinado número de iteraciones, el resultado que producirá siempre será muy bueno.

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 implican ajustes en la maquinaria. Concretamente:

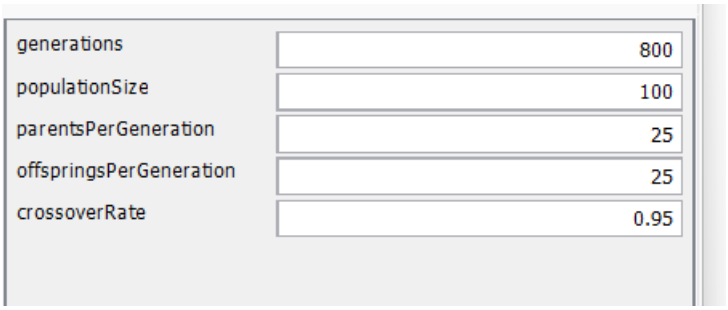
- Si en el mismo tramo (1, 2 o 3) se fabrican P2, P3 y P7, el beneficio se aumenta en 5€.
- No queremos que en el mismo tramo (1, 2 o 3) se fabriquen P10 y P12, por lo que debemos penalizar mucho esta situación.

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).

En el *evaluator*:

```
package proyectoelectricidad;

import java.util.ArrayList;
import org.opt4j.core.Objectives;
import org.opt4j.core.Objective.Sign;
import org.opt4j.core.problem.Evaluator;

public class Ejercicio2 implements Evaluator<ArrayList<Integer>> {
	public Objectives evaluate(ArrayList<Integer> phenotype) {
		double beneficio = 0;
		int productos = 0;
		double[] potenciaActual = new double[DatosElectricidad.NUM_TRAMOS + 1];
		int tramo;
		int bono = 10;
		boolean existeBono = false;
		int counter = 0;
		int tramo1 = -1;
		int tramo2 = -1;

		for (int i = 0; i < phenotype.size(); ++i) {
			tramo = phenotype.get(i);

			if (i == 0) {
				tramo1 = tramo;
			}
			if (i == 1) {
				tramo2 = tramo;
			}

			// Condicion tener suficiente potencia en ese tramo
			if (tramo != 0 && potenciaActual[tramo]
					+ DatosElectricidad.consumo[i] >
DatosElectricidad.potenciaMaximaTramo[tramo]) {
				// Si no hay suficiente potencia se deja para la siguiente jornada laboral
				// penalizando su beneficio
				phenotype.set(i, 0);
				tramo = 0;
			}

			// Para aquellos productos que no se han dejado para la proxima jornada laborar
			if (tramo > 0) {
				if (tramo1 > 0 && (i == 1 || i == 2 || i == 3) && tramo1 == tramo) {
					beneficio -= Integer.MAX_VALUE;
				}

				if (tramo2 > 0 && (i == 2 || i == 3 || i == 7) && tramo2 == tramo) {
					++counter;
				}
```

```
                                potenciaActual[tramo] += DatosElectricidad.consumo[i];
                                if (tramo == 1 || tramo == 2) {
                                        ++productos;
                                }
                                beneficio += DatosElectricidad.beneficio[i] - (DatosElectricidad.consumo[i]
                                                * DatosElectricidad.horasTramo[tramo] *
DatosElectricidad.preciokWhTramo[tramo]);
                        }
                }

                for (int i = 1; i < DatosElectricidad.NUM_TRAMOS; ++i) {
                        tramo = phenotype.get(i);
                        if (potenciaActual[tramo] < DatosElectricidad.potenciaMaximaTramo[i] * 0.9) {
                                existeBono = true;
                        }
                }

                if (existeBono) {
                        beneficio += bono;
                }

                if (counter == 2) {
                        beneficio += 5;
                }

                Objectives objectives = new Objectives();
                objectives.add("Beneficio-MAX", Sign.MAX, beneficio);
                objectives.add("Productos-MAX", Sign.MAX, productos);
                return objectives;
        }
}
```



Individuo Beneficio Unidades

1 [2, 1, 1, 1, 2, 3, 1, 2, 2, 2, 3, 2, 3, 1, 3, 1, 0, 3, 3, 2] 66.012 13

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)	24	20
Precio del kWh (€)	0.11	0.18

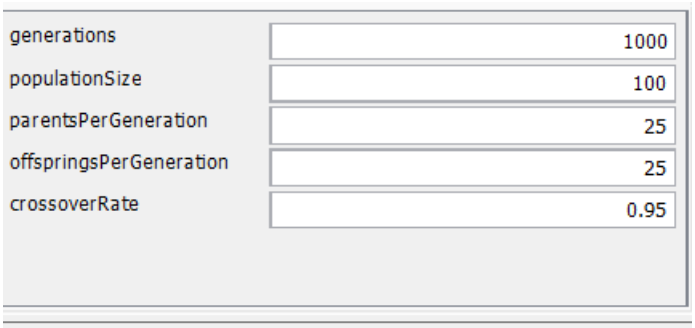
Además, deseamos poder fabricar 2 productos más (P21 y P22) con la siguiente información:

	P21	P22
Consumo (kWh)	3.5	4.7
Beneficio (€)	4	6

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).

En ***DatosElectricidad.java:***

```
package proyectoelectricidad;

public class DatosElectricidad {

	public static final int NUM_TRAMOS = 4;
	public static final int NUM_PEDIDOS = 20;

	// 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, 3.5, 4.7};

	// 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, 4, 6};

	// 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, 24, 20 };

	// precio del kWh para cada tramo (el tramo 0 es ficticio)
	public static final double[] precioKWhTramo = { 0, 0.26, 0.18, 0.11, 0.18 };

}
```



Individuo	Beneficio	Unidades
1 [2, 1, 1, 1, 2, 2, 3, 2, 1, 2, 3, 4, 2, 1, 4, 1, 3, 3, 3, 2]	77.404	13
2 [3, 2, 2, 1, 2, 4, 3, 2, 3, 3, 3, 2, 4, 2, 4, 2, 4, 4, 3, 4]	85.314	8
3 [2, 1, 1, 1, 2, 4, 3, 4, 2, 2, 3, 4, 4, 2, 4, 2, 3, 3, 3, 2]	82.708	10
4 [2, 1, 1, 1, 2, 4, 3, 2, 1, 2, 3, 4, 2, 1, 4, 2, 3, 3, 3, 2]	79.456	12

5 [3, 2, 2, 1, 3, 4, 3, 2, 3, 3, 3, 2, 4, 2, 4, 2, 4, 4, 3, 4] 86.064 7
6 [3, 2, 2, 4, 3, 2, 3, 2, 3, 3, 3, 2, 4, 2, 4, 4, 4, 4, 3, 4] 86.324 6
7 [2, 1, 3, 1, 2, 3, 3, 1, 2, 3, 4, 4, 2, 1, 2, 2, 3, 4, 3, 2] 81.362 11
8 [2, 1, 3, 1, 2, 3, 3, 1, 2, 3, 4, 4, 4, 2, 2, 4, 3, 4, 3, 2] 83.91399999999999 9

4. (1 punto) Explica razonadamente **(no es necesario implementar nada**) cuál sería el mejor genotipo si se desearan distribuir tantos turnos como productos a fabricar, sin repetir ninguno. Es decir, cada producto se fabrica en un único turno y en cada turno se fabrica un único producto.

El PermutationGenotype sería una buena opción, ya que distribuiría los turnos y los entregaría por pantalla de una manera ordenada y sin repeticiones.