Práctica Individual 3

Problema Número 2

Análisis y Diseño de Datos y Algoritmos / Estructura de Datos y Algoritmos

Grado de Ingeniería de Software

Curso 2º

Luis Candelario Luna (luisitiluna@hotmail.com)

Tutor: María Luisa Parody

Índice

1 Enunciado y Cuestiones a resolver

Una empresa de transportes desea enviar mercancía en diferentes vagones de tren. Cada vagón tiene un peso y un valor asociado a su contenido. Una locomotora puede transportar un peso menor o igual al peso máximo establecido, es decir, los vagones transportados no deben superar dicho peso. Los vagones pueden requerir una vigilancia especial o que se les suministre electricidad. Sólo se puede asociar un vagón con vigilancia a la locomotora, y un máximo de 2 vagones que requieran electricidad. El objetivo es encontrar la lista de vagones, que cumpliendo las restricciones comentadas, permita maximizar el valor de los vagones transportados. A continuación se describe un ejemplo en el que para cada vagón se indica el peso del vagón, el valor, si requiere vigilancia o electricidad, y el número máximo de unidades disponibles.

Para un peso máximo de 20, pueden enviarse un contenedor ID 1, ID3 y otro ID5 que daría un valor de 24, pero la mejor opción, que cumpla las restricciones, es un vagón ID 5 y dos ID 3, lo que daría un valor de 28.

SE PIDE(*)(**):

(1) Resolver el problema por PL o PLI, para ello:

a. Indique razonadamente si es adecuado usar PL ó PLI.

b. Indique cuál es la especificación del problema (que debe incluir la función objetivo, restricciones y tipo de las variables) de forma análoga a como se ha hecho en clases de teoría y prácticas. Justifique por qué ha incluido cada variable y cada restricción.

c. Implemente los métodos que aparecen como //TODO en el proyecto entregado. Adicionalmente, puede añadir tantas clases, métodos y/o atributos como considere necesarios. Tenga en cuenta que debe dar una implementación general que genere la solución requerida para cualquier problema de entrada, y no sólo para el escenario concreto que se proporciona en este enunciado.

(2) Resolver el problema mediante AG, para ello:

a. ¿Qué tipo o tipos de cromosomas son los más adecuados para resolver el problema y por qué?

b. Implemente los métodos que aparecen como //TODO en el proyecto entregado. Adicionalmente, puede añadir tantas clases, métodos y/o atributos como considere necesarios.

c. Complete el test de prueba e indique qué solución obtiene para el problema propuesto en el enunciado. Los datos de los vagones se facilitan en el fichero Vagones.txt

2 Seguimiento

2.1 Control de seguimiento y tutorías

Fecha	Profesor	Detalles y dudas resueltas
14/03/2017	Luisa Parody	- Problemas al resolver la declaración de variables, no conseguía mostrar el formato exacto [SOLUCIONADO}.

2.2 Portafolio

Fecha	Evolución del Problema
10/03/2017	- Descarga del proyecto y compresión del enunciado
12/03/2017	- Realización de la función objetivo, restricciones y tipo de las variables en la clase VagonesPLI
15/03/2017	- Realización de la clase TestVagonesPLI y comprobación de los resultados.
18/03/2017	-Implementación de los métodos necesarios de la clase ProblemaVagonesAG y de la clase TestVagonesAG
10/03/2017	-Comprobación de los resultados obtenidos al ejecutar el algoritmo.

(1) A. Es más adecuado usar PLI porque nuestras variables tienen que ser números enteros. Nuestras variables representan la cantidad de vagones que vamos a coger de cada uno, no podemos coger porciones de vagones por lo tanto utilizamos PLI.

(2) A. El tipo de cromosoma que hemos utiliza es el Cromosoma Index ya que nos muestra una lista con la cantidad de vagones de cada tipo que vamos a coger. No podemos utilizar el cromosoma Binary porque el enunciado del problema no nos exige si hemos cogido el vagón de un tipo o no sino que nos exige cuantos vagones de cada tipo debemos de escoger.

3 Implementación

Clase VagonesPLI

```
public class VagonesPLI {
	// TODO
	public static String getConstraints() {
		String res = "max:";

		// FUNCION OBJETIVO
		List<Vagon> lista = ProblemaVagones.listaVagones;
		for (Vagon v : lista) {
			res += " +" + v.getValor() + "x" + v.getCodigo();
		}

		res += ";\n";


		// EXCEPCION UNIDADES DISPONIBLES

		for (Vagon v : lista) {
			res += "x" + v.getCodigo() + " <= " +
v.getNumMaxDeUnidades() + ";\n";
		}

		// EXCEPCION PESO

		for (Vagon v : lista) {
			res += " + " + v.getPeso() + "x" + v.getCodigo();
		}
		res += " <=" + ProblemaVagones.pesoTotal+";\n";

		// EXCEPCION VIGILANCIA

		for (Vagon v : lista) {
			if (v.isVigilancia()) {
				res += " + x" + v.getCodigo();
			}
```

```
            }
            res += " <=1;\n";


            // EXCEPCION ELECTRICIDAD

            for (Vagon v : lista) {
                  if (v.isElectricidad()) {
                        res += " + x" + v.getCodigo();
                  }
            }
            res += " <=2;\n";

            // DECLARACION DE VARIABLES

            res += "int ";

            for (Vagon v : lista) {
                  if (v.getCodigo() ==
ProblemaVagones.listaVagones.size()-1) {
                        res += "x"+v.getCodigo()+";";
                  } else {
                        res += "x" + v.getCodigo() + ",";
                  }

            }

            return res;

      }


}
```

Clase ProblemasVagonesAG

```
public class ProblemaVagonesAG implements
ProblemaAGIndex<SolucionVagones>{


      public Integer getObjectsNumber() {
            return ProblemaVagones.listaVagones.size();
      }



      public Double fitnessFunction(IndexChromosome cr) {


            Integer numeroVagonesElectricidad = 0;
```

```
            Integer numeroVagonesVigilancia= 0;
            Double fitness=0.0;
            Integer indice= 0;
            Double peso=0.0;
            Integer indice1 = 0;
            Double valor=0.0;

            for(Integer i:cr.decode()){

      if(ProblemaVagones.listaVagones.get(indice1).isElectricidad()){

                  numeroVagonesElectricidad += i;
            }

if(ProblemaVagones.listaVagones.get(indice1).isVigilancia()){
                numeroVagonesVigilancia += i;
                }
                indice1++;
            }

            if(numeroVagonesElectricidad>2 || numeroVagonesVigilancia
>1){
                  fitness= -
1000000.0*(numeroVagonesElectricidad*numeroVagonesVigilancia);
                  return fitness;

            }else{

            for(Integer numeroVecesVagon: cr.decode()){

                  valor += numeroVecesVagon*
ProblemaVagones.listaVagones.get(indice).getValor();

                  peso += numeroVecesVagon*
ProblemaVagones.listaVagones.get(indice).getPeso();

                  indice++;

            }
            if(peso>ProblemaVagones.pesoTotal){
                  fitness= -peso*(peso*peso);
            }else{
                  fitness=valor;
            }
            return fitness;
            }}


      public SolucionVagones getSolucion(IndexChromosome cr) {
            Map<Vagon,Integer> map= new HashMap<Vagon,Integer>();

            List<Integer> ls = cr.decode();
            for (int i=0; i< ls.size();i++) {
                  Vagon v = ProblemaVagones.listaVagones.get(i);
                  map.put(v, ls.get(i));
            }
```

```
            return SolucionVagones.create(map);
      }


      public  Integer getMax(int index){
            return
ProblemaVagones.listaVagones.get(index).getNumMaxDeUnidades();

      }
      public List<Integer> getNormalSequence() {
            return IntStream.range(0,getObjectsNumber())
                        .boxed()
                        .map(x->Lists2.nCopias(getMax(x),x).stream())
                        .flatMap(x->x)
                        .collect(Collectors.toList());
      }

}
```

4 Test y Pruebas

Clase TestVagonesPLI

```
public class TestVagonesPLI {

      public static void main(String[] args) {
            ProblemaVagones.create("Vagones.txt");
            ProblemaVagones.pesoTotal = 20;
            System.out.println("------");
            System.out.println("Peso total: " +
ProblemaVagones.pesoTotal);
            System.out.println("------");
            System.out.println("Vagones:\n" +
ProblemaVagones.listaVagones);
            System.out.println("------");

            String r = VagonesPLI.getConstraints();
            // TODO
            AlgoritmoPLI a =
Algoritmos.createPLI("ficheros/restricciones.txt");
            a.setConstraints(r);
            a.ejecuta();
            System.out.println("Especificación LPSolve:");
            System.out.println(r);
            System.out.println("Objetivo Solución Óptima = "
+a.getObjetivo());
            System.out.println("Solución Óptima = ");
            for(int i=0; i<=a.getSolucion().length-1; i++){
```

```
                    System.out.println("x" + i + " = " +
a.getSolucion(i));
                    }
      }

}
```

*Salida Obtenida**

```
------
Peso total: 20
------
Vagones:
[Vagón 0, Vagón 1, Vagón 2, Vagón 3, Vagón 4, Vagón 5]
------
Especificación LPSolve:
max: +1.0x0 +3.0x1 +4.0x2 +7.0x3 +3.0x4 +14.0x5;
x0 <= 4;
x1 <= 5;
x2 <= 2;
x3 <= 3;
x4 <= 2;
x5 <= 2;
 + 1x0 + 2x1 + 4x2 + 5x3 + 3x4 + 9x5 <=20;
 + x2 + x5 <=1;
 + x0 + x1 + x2 + x3 <=2;
int x0,x1,x2,x3,x4,x5;
Objetivo Solución Óptima = 28.0
Solución Óptima =
x0 = 0.0
x1 = 0.0
x2 = 0.0
x3 = 2.0
x4 = 0.0
x5 = 1.0
```

Clase TestVagonesAG

```
public class TestVagonesAG {

      // TODO
      public static void main(String[] args){

            AlgoritmoAG.ELITISM_RATE  = 0.30;
            AlgoritmoAG.CROSSOVER_RATE = 0.8;
            AlgoritmoAG.MUTATION_RATE = 0.7;
            AlgoritmoAG.POPULATION_SIZE = 100;
            StoppingConditionFactory.NUM_GENERATIONS = 400;
            StoppingConditionFactory.SOLUTIONS_NUMBER_MIN = 10;
            StoppingConditionFactory.FITNESS_MIN = 100.0;
```

```
            StoppingConditionFactory.stoppingConditionType =
StoppingConditionType.GenerationCount;

            ProblemaVagones.create("Vagones.txt");
            ProblemaVagones.pesoTotal = 20;
            System.out.println("------");
            System.out.println("Peso total: " +
ProblemaVagones.pesoTotal);
            System.out.println("------");
            System.out.println("Vagones:\n" +
ProblemaVagones.listaVagones);
            System.out.println("------");
            System.out.println("Solución :");
            ProblemaVagonesAG p = new ProblemaVagonesAG();
            // TODO
            AlgoritmoAG a =
Algoritmos.createAG(ChromosomeType.IndexRange, p);
            a.ejecuta();
            a.getBestFinal();
            IndexChromosome cromosoma=
ChromosomeFactory.asIndex(a.getBestFinal());
            SolucionVagones v = p.getSolucion(cromosoma);

            //System.out.println(a.getBestFinal());
            System.out.println("Mejor solución: "+ v);
            System.out.println("Fitness de la mejor solución: "+
a.getBestFinal().getFitness());
            System.out.println("------");
      }

}
```

*Salida Obtenida**

```
------
Peso total: 20
------
Vagones:
[Vagón 0, Vagón 1, Vagón 2, Vagón 3, Vagón 4, Vagón 5]
------
Solución :
Mejor solución: SolucionVagones [vagones={Vagón 1=0, Vagón 2=0, Vagón
3=2, Vagón 4=0, Vagón 5=1, Vagón 0=0}, peso=19, valor=28.0]
Fitness de la mejor solución: 28.0
------
```

*A veces en la salida obtenida tras ejecutar la clase TestProblemaVagonesAG aparece El valor de Fitness de la mejor solución es 27.0 pero tras haberme comunicado con el profesor me ha confirmado que es problema del Algoritmo y no del código que he programado.