

Tabla de contenido

Ejercicio1	2
DatosEjercicio1.java	2
SolucionEjercicio1.java.....	5
Ejercicio1AG.java	6
Ejercicio1.Isi	8
Ejercicio1PLE.java.....	9
TestEj1AGRange.java	10
Resultados PLE	11
Ejercicio2	15
DatosEjercicio2.java	15
SolucionEjercicio2.java.....	17
Ejercicio2AG.java	18
Ejercicio2PLE.java.....	20
Ejercicio2.Isi	21
TestEj2AGRange.java	22
Resultados AG	23
Resultados PLE	24
Ejercicio3	27
DatosEjercicio3.java	27
SolucionEjercicio3.java.....	30
Ejercicio3AG.java	32
Ejercicio3PLE.java.....	35
Ejercicio3.Isi	36
TestEj3AGRange.java	37
Resultados AG	38
Resultados PLE	39
Ejercicio4	43
DatosEjercicio4.java	43
SolucionEjercicio4.java.....	45
Ejercicio4AG.java	47
TestEj4AGRange.java	49
Resultados AG	50

Ejercicio1

DatosEjercicio1.java

```
package _datos;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Set;
import java.util.stream.Collectors;

import us.lsi.common.Files2;
import us.lsi.common.String2;

public class DatosEjercicio1 {

    public static List<Tipo> tipos;
    public static record Tipo(String Nombre_Tipo, Integer kgdisponibles, Integer id) {
        public Tipo of(String Nombre_Tipo, Integer kgdisponibles, Integer id) {
            return new Tipo(Nombre_Tipo, kgdisponibles, id);
        }

        public static Tipo ofFormat(String linea) {
            String[] formato = linea.split(":");
            String Nombre_Tipo = formato[0];
            Integer kgdisponibles =
Integer.parseInt(formato[1].replace("kgdisponibles=", "").replace(";", "").trim());
            Integer id = Integer.parseInt(formato[2].replace("C", ""));
            return new Tipo(Nombre_Tipo, kgdisponibles, id);
        }
    }

    public static List<Variedad> variedades;
    public static record Variedad(String nombre, Integer id, Double beneficio, Map<String,
Double> porcentaje) {
        public Variedad of(String nombre, Integer id, Double beneficio, Map<String,
Double> porcentaje) {
            return new Variedad(nombre, id, beneficio, porcentaje);
        }

        public static Variedad ofFormat(String linea) {
            String[] formato = linea.split("->");
            String nombre = formato[0].trim();
            String segundaParte = formato[1].trim();
            String[] segundaPartePars = segundaParte.split(";");
            Double beneficio =
Double.parseDouble(segundaPartePars[0].split("=")[1].trim());
            Map<String, Double> porcentaje =
Arrays.stream(segundaPartePars[1].split("=")[1].split(","))
                .map(pair -> pair.split(":"))
                .collect(Collectors.toMap(
                    keyValue -> keyValue[0].replace("(", "").replace(")", ""),
                    keyValue -> Double.parseDouble(keyValue[1].replace("(",
"".replace(")", ""))
                ));

            Integer id = null;
        }
    }
}
```

```

        return new Variedad(nombre, id, beneficio, porcentaje);
    }
}

public static void iniDatos(String fichero) {
    List<String> lineas = Files2.LinesFromFile(fichero);
    tipos = lineas.stream().filter(l -> l.startsWith("C")).map(x ->
Tipo.ofFormat(x)).toList();
    variedades = lineas.stream().filter(l -> l.startsWith("P")).map(x ->
Variedad.ofFormat(x)).toList();
    toConsole();
}

//.....
//head section del .lsi

public static Integer getTipos() {
    return tipos.size();
}

public static Integer getVariedades() {
    return variedades.size();
}

public static Double getBeneficio(Integer i) {
    return variedades.get(i).beneficio;
}
public static Double getPorcentajeVariedad(Integer i, Integer j) {
    Set<String> setTipos = variedades.get(i).porcentaje.keySet();
    String aux = tipos.stream().filter(x -> tipos.indexOf(x) ==
j).map(Tipo::Nombre_Tipo).findFirst().orElse(null);
    return setTipos.contains(aux)?variedades.get(i).porcentaje().get(aux):0.;
}
public static Integer getKgDisponibles(Integer i) {
    return tipos.get(i).kgdisponibles;
}
public static Integer getKgDisponiblesMax() {
    return tipos.stream().map(Tipo::kgdisponibles).reduce((a, b) -> a > b?a:b).get();
}

public static Double getBounds(Integer i, Integer j) {
    Tipo tipo = tipos.get(i);
    Double porcentaje = getPorcentajeVariedad(i, j);
    return tipo.kgdisponibles()/porcentaje;
}

//.....
public static List<Tipo> getListaTipos(){
    return tipos;
}
public static List<Variedad> getListaVariedades(){
    return variedades;
}

public static void toConsole() {
    //String2.toConsole("Conjunto de Entrada Tipos: %s\nConjunto de Entrada
Variedades: %d", tipos, variedades);
    System.out.println(tipos);
    System.out.println(variedades);
}

```

```
// Test de la lectura del fichero
public static void main(String[] args) {
    iniDatos("ficheros/Ejercicio1DatosEntrada1.txt");
}
}
```

SolucionEjercicio1.java

```

package _soluciones;

import java.util.List;

import _datos.DatosEjercicio1;
import _datos.DatosEjercicio1.Variedad;
import us.lsi.common.List2;

public class SolucionEjercicio1 {

    public static SolucionEjercicio1 of_Range(List<Integer> ls) {
        return new SolucionEjercicio1(ls);
    }

    private Double beneficio; //beneficio que tenemos
    private List<Variedad> variedades; //variedades que tenemos
    private List<Integer> solucion; //cantidad que cogemos de cada variedad

    private SolucionEjercicio1() {
        beneficio = 0.;
        solucion = List2.empty();
        variedades = List2.empty();
    }

    private SolucionEjercicio1(List<Integer> ls) {
        beneficio = 0.;
        solucion = List2.of();
        variedades = List2.empty();
        for(int i=0; i<ls.size(); i++) {
            if(ls.get(i)>0) {
                Integer e = ls.get(i); //x[i]
                Double v = DatosEjercicio1.getBeneficio(i); //beneficio
                beneficio += v * e; //beneficio += beneficio[i] * x[i]
                variedades.add(DatosEjercicio1.variedades.get(i));
                solucion.add(e);
            }
        }
    }

    public static SolucionEjercicio1 empty() {
        return new SolucionEjercicio1();
    }

    public static SolucionEjercicio1 create(List<Integer> ls) {
        return new SolucionEjercicio1(ls);
    }

    @Override
    public String toString() {
        // int error = Math.abs(DatosEjercicio1.getSuma() - suma);
        // String e = error<1? "": String.format("Error = %d", error);
        return String.format("Solucion = %s; Tamaño solucion = %d; beneficio total = %f;",
            solucion, solucion.size(), beneficio);
    }
}

```

Ejercicio1AG.java

```

package ejercicio1;

import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.util.Map;

import _datos.DatosEjercicio1;
import _datos.DatosEjercicio1.Tipo;
import _datos.DatosEjercicio1.Variedad;
import _soluciones.SolucionEjercicio1;
import us.lsi.ag.ValuesInRangeData;
import us.lsi.ag.agchromosomes.ChromosomeFactory.ChromosomeType;
import us.lsi.bufete.datos.TipoSolucion;
import us.lsi.common.List2;

public class Ejercicio1AG implements ValuesInRangeData<Integer, SolucionEjercicio1> {

    public Ejercicio1AG(String linea) {
        DatosEjercicio1.iniDatos(linea);
    }

    @Override
    public Integer size() {
        // TODO Auto-generated method stub
        return DatosEjercicio1.getVariedades();
    }

    @Override
    public ChromosomeType type() {
        // TODO Auto-generated method stub
        return ChromosomeType.Range;
    }

    @Override
    public Double fitnessFunction(List<Integer> cromosoma_variedad) {
        // TODO Auto-generated method stub
        Double goal = 0.;
        for (int i = 0; i < cromosoma_variedad.size(); i++) {
            goal+= cromosoma_variedad.get(i) * DatosEjercicio1.getBeneficio(i);
        }

        //Fitness Maximo
        Double fM = 0.;
        for (int i = 0; i < cromosoma_variedad.size(); i++) {
            fM+= this.max(i)*2*DatosEjercicio1.getBeneficio(i);
        }

        //Restricciones
        Integer restriccion = 0;

        for (int cafe = 0; cafe < DatosEjercicio1.tipos.size(); cafe++) {
            Boolean compruebo = false;
            Double suma = 0.;
            for(int variedad = 0; variedad < cromosoma_variedad.size();variedad++) {
                suma += cromosoma_variedad.get(variedad) *
DatosEjercicio1.getPorcentajeVariedad(variedad, cafe);
            }
            compruebo = suma <= DatosEjercicio1.tipos.get(cafe).kgdisponibles();
        }
    }

```

```

        if(!compruebo) {
            System.out.println("");
        }
        restriccion+=compruebo?0:1;
    }

    return 0.0 + goal - (restriccion * fM);
}

@Override
public SolucionEjercicio1 solucion(List<Integer> value) {
    System.out.println(value); //esto es solo para hacer pruebas
    return SolucionEjercicio1.create(value);
}

@Override
public Integer max(Integer i) {
    // TODO Auto-generated method stub
    List<Double> aux = List2.empty();
    for (Tipo v : DatosEjercicio1.tipos) {
        Integer cafe_i = DatosEjercicio1.tipos.indexOf(v);
        Integer kg_cafe_i = DatosEjercicio1.getKgDisponibles(cafe_i);
        Double porcentaje_cafe_i_var_j = DatosEjercicio1.getPorcentajeVariedad(i,
cafe_i);

        aux.add((kg_cafe_i/porcentaje_cafe_i_var_j)+1);
    }
    return aux.stream().reduce((a,b) -> a<b?a:b).get().intValue();
}

@Override
public Integer min(Integer i) {
    // TODO Auto-generated method stub
    return 0;
}
}

```

Ejercicio1.lsi

head section

```
Integer getTipos()
Integer getVariedades()
Integer getKgDisponiblesMax()

Integer getKgDisponibles(Integer i)
Double getBeneficio(Integer i)
Double getPorcentajeVariedad(Integer i, Integer j)
Double getBounds(Integer i, Integer j)

Integer n = getTipos()
Integer m = getVariedades()
```

goal section

```
max sum(getBeneficio(i) x[i] , i in 0 .. m)
```

constraints section

```
sum(getPorcentajeVariedad(i,j) x[i], i in 0 .. m) <= getKgDisponibles(j), j in 0 .. n
```

bounds section

```
x[i] <= getBounds(i,j), i in 0 .. m, j in 0 .. n
```

int

```
x[i], i in 0 .. m
```


Ejercicio1PLE.java

```

package ejercicio1;

import java.io.IOException;
import java.util.List;
import java.util.Set;

import _datos.DatosEjercicio1;
import _datos.DatosEjercicio1.Tipo;
import _datos.DatosEjercicio1.Variedad;
import us.lsi.common.String2;
import us.lsi.gurobi.GurobiLp;
import us.lsi.gurobi.GurobiSolution;
import us.lsi.solve.AuxGrammar;

public class Ejercicio1PLE {
    public static void ejemplo1_model() throws IOException {
        //Leer Datos de entrada
        DatosEjercicio1.iniDatos("ficheros/Ejercicio1DatosEntrada1.txt");

        //      System.out.println("variedades:::" + variedades );
        //      tipos = DatosEjercicio1.getListaTipos();
        //      variedades = DatosEjercicio1.getListaVariedades();

        //si cambia el fichero de datos de entrada, cambiar tambien el nº del .lp para no
        //sobreescribirlo
        //Pasar de LSI a Gurobi
        AuxGrammar.generate                                     //generar un archivo gurobi
            (DatosEjercicio1.class,                             //la clase que utilizo
            "lsi_models/ejercicio1.lsi",                         //de donde saco el modelo
            "gurobi_models/Ejercicio1-1.lp"); //donde lo llevo

        //Usar Gurobi
        GurobiSolution solution = GurobiLp.gurobi("gurobi_models/Ejercicio1-1.lp");
        //Interpretar solucion
        String2.toConsole("\nSolucion PLE: %s", solution.toString((s,d)-
        >d>0.).substring(2));

        //      Locale.setDefault(new Locale("en", "US"));
        //      System.out.println(solution.toString((s,d)->d>0.));
    }

    public static void main(String[] args) throws IOException {
        ejemplo1_model();
    }
}

```

TestEj1AGRange.java

```

package ejercicio1;

import java.util.List;
import java.util.Locale;

import _soluciones.SolucionEjercicio1;
import us.lsi.ag.agchromosomes.AlgoritmoAG;
import us.lsi.ag.agstopping.StoppingConditionFactory;

public class TestEj1AGRange {

    public static void main(String[] args) {
        Locale.setDefault(new Locale("en", "US"));

        AlgoritmoAG.ELITISM_RATE = 0.10;
        AlgoritmoAG.CROSSOVER_RATE = 0.95;
        AlgoritmoAG.MUTATION_RATE = 0.8;
        AlgoritmoAG.POPULATION_SIZE = 1000;

        StoppingConditionFactory.NUM_GENERATIONS = 1000;
        StoppingConditionFactory.stoppingConditionType =
        StoppingConditionFactory.StoppingConditionType.GenerationCount;

        Ejercicio1AG p = new Ejercicio1AG("ficheros/Ejercicio1DatosEntrada" + i +
        ".txt");

        AlgoritmoAG<List<Integer>, SolucionEjercicio1> ap = AlgoritmoAG.of(p);
        ap.ejecuta();
        System.out.println("=====");
        System.out.println(ap.bestSolution());
        System.out.println("=====\\n");
    }
}

```

Resultados PLE

Maximize

$$20.0 \ x_0 + 10.0 \ x_1 + 5.0 \ x_2$$

Subject To

$$a0: 0.5 \ x_0 + 0.0 \ x_1 + 0.0 \ x_2 \leq 5$$

$$a1: 0.4 \ x_0 + 0.0 \ x_1 + 0.0 \ x_2 \leq 4$$

$$a2: 0.1 \ x_0 + 0.0 \ x_1 + 0.0 \ x_2 \leq 1$$

$$a3: 0.0 \ x_0 + 0.2 \ x_1 + 0.0 \ x_2 \leq 2$$

$$a4: 0.0 \ x_0 + 0.8 \ x_1 + 0.0 \ x_2 \leq 8$$

$$a5: 0.0 \ x_0 + 0.0 \ x_1 + 1.0 \ x_2 \leq 1$$

Bounds

$$x_0 \leq 10.0$$

$$x_0 \leq 12.5$$

$$x_0 \leq 50.0$$

$$x_0 \leq \text{Infinity}$$

$$x_0 \leq \text{Infinity}$$

$$x_0 \leq \text{Infinity}$$

$$x_1 \leq \text{Infinity}$$

$$x_1 \leq \text{Infinity}$$

$$x_1 \leq \text{Infinity}$$

$$x_1 \leq 20.0$$

$$x_1 \leq 5.0$$

$$x_1 \leq \text{Infinity}$$

$$x_2 \leq \text{Infinity}$$

$$x_2 \leq \text{Infinity}$$

$$x_2 \leq \text{Infinity}$$

$$x_2 \leq \text{Infinity}$$

$$x_2 \leq \text{Infinity}$$

$$x_2 \leq 1.0$$

General

$$x_0 \ x_1 \ x_2$$

End

Maximize

$$20.0 \ x_0 + 10.0 \ x_1 + 80.0 \ x_2$$

Subject To

$$a0: 0.2 \ x_0 + 0.0 \ x_1 + 0.4 \ x_2 \leq 11$$

$$a1: 0.4 \ x_0 + 0.3 \ x_1 + 0.0 \ x_2 \leq 9$$

$$a2: 0.0 \ x_0 + 0.7 \ x_1 + 0.0 \ x_2 \leq 7$$

$$a3: 0.0 \ x_0 + 0.0 \ x_1 + 0.6 \ x_2 \leq 12$$

$$a4: 0.4 \ x_0 + 0.0 \ x_1 + 0.0 \ x_2 \leq 6$$

Bounds

$$x_0 \leq 55.0$$

$$x_0 \leq 27.5$$

$$x_0 \leq \text{Infinity}$$

$$x_0 \leq \text{Infinity}$$

$$x_0 \leq 27.5$$

$$x_1 \leq \text{Infinity}$$

$$x_1 \leq 30.0$$

$$x_1 \leq 12.857142857142858$$

$$x_1 \leq \text{Infinity}$$

$$x_1 \leq \text{Infinity}$$

$$x_2 \leq 17.5$$

$$x_2 \leq \text{Infinity}$$

$$x_2 \leq \text{Infinity}$$

$$x_2 \leq 11.666666666666668$$

$$x_2 \leq \text{Infinity}$$

General

$$x_0 \ x_1 \ x_2$$

End


```
x_3 <= Infinity
x_3 <= 25.0
x_4 <= Infinity
x_4 <= Infinity
x_4 <= 75.0
x_4 <= Infinity
x_4 <= Infinity
x_4 <= Infinity
x_4 <= Infinity
x_4 <= 50.0
x_4 <= Infinity
x_4 <= Infinity
x_5 <= 210.0
x_5 <= Infinity
x_5 <= Infinity
x_5 <= Infinity
x_5 <= 140.0
x_5 <= 140.0
x_5 <= Infinity
x_5 <= Infinity
x_5 <= 210.0
x_5 <= Infinity
```

General

```
x_0 x_1 x_2 x_3 x_4 x_5
```

End

Ejercicio2

DatosEjercicio2.java

```
package _datos;

import java.util.HashSet;
import java.util.List;
import java.util.Set;
import java.util.stream.Collectors;

import _datos.DatosEjercicio1.Tipo;
import _datos.DatosEjercicio1.Variedad;
import _datos.DatosEjercicio2.Curso;
import us.lsi.common.Files2;
import us.lsi.common.List2;
import us.lsi.common.String2;

public class DatosEjercicio2 {

    private static int id_aux = 0;
    public static Integer maxCentros;
    public static List<Curso> cursos;
    public record Curso(Integer id, Set<Integer> tematicas, Double coste, Integer centro) {
        public Curso of(Integer id, Set<Integer> tematicas, Double coste, Integer centro)
    {
        return new Curso(id, tematicas, coste, centro);
    }

    public static Curso ofFormat(String linea) {
        String[] formato = linea.split(":");
        Integer id = id_aux++;
        Set<Integer> tematicas = auxiliar(formato[0]);
        Double coste = Double.parseDouble(formato[1]);
        Integer centro = Integer.parseInt(formato[2]);
        return new Curso(id,tematicas, coste, centro);
    }

    }

    public static Set<Integer> auxiliar(String formato){
        Set<Integer> res = new HashSet<>();
        String[] aux = formato.replace("{", "").replace("}", "").split(",");
        for (String auxi : aux) {
            res.add(Integer.parseInt(auxi));
        }
        return res;
    }

    public static void iniDatos(String fichero) {
        String[] v = Files2.linesFromFile(fichero).get(0).split(":");
        List<String> lineas = Files2.linesFromFile(fichero);
        maxCentros = lineas.stream().filter(l -> l.contains("Max_Centros")).map(x ->{
            String[] parts = x.split("=");
            return Integer.parseInt(parts[1].trim());
        }).findFirst().orElse(null);
        cursos = lineas.stream().filter(l -> l.startsWith("{")).map(x ->
Curso.ofFormat(x)).toList();
        //toConsole();
        System.out.println(cursos);
        System.out.println("Max_Centros = " + maxCentros);
    }
}
```

```

}

//.....
//head section del .lsi
public static Integer getCursos() { //nº de cursos
    return cursos.size();
}
public static Integer getTematicas() { //nº de tematicas
    Set<Integer> aux = new HashSet<>();
    cursos.stream().map(Curso::tematicas).forEach(x -> aux.addAll(x));
//    System.out.println("Hay " + (int) aux.stream().distinct().count() + " tematicas");
//    System.out.println(aux.stream().distinct().collect(Collectors.toList()));
    return (int) aux.stream().distinct().count();
}
public static List<Integer> getListaTematicas(){
    return cursos.stream().flatMap(x ->
x.tematicas().stream()).distinct().collect(Collectors.toList());
}
public static Integer getCentros() { //nº de centros
    return getListaCentros().size();
}
public static Integer getCentrosDiferentes() { //maxCentros
    return maxCentros;
}
public static Double getPrecioInscripcion(Integer i) { //coste de un curso
    return cursos.stream().filter(c -> c.id()==i).findFirst().map(Curso::coste).get();
}
public static Integer seleccionaTematica(Integer i, Integer j) { //devuelve 1 si se
selecciona la tematica j
    return cursos.get(i).tematicas().contains(getListaTematicas().get(j))?1:0;
}
public static List<Integer> getListaCentros() {
    return cursos.stream().map(Curso::centro).distinct().collect(Collectors.toList());
}
public static Integer seleccionaCentro(Integer i, Integer j) { //devuelve 1 si sel
centro j ofrece el curso i
    return cursos.get(i).centro().equals(getListaCentros().get(j))?1:0;
}

public static double maximoCoste() {
    return cursos.stream().map(Curso::coste).reduce((a, b) -> a>b?a:b).get();
}
//.....

public static List<Curso> getListaCursos(){
    return cursos;
}

// public static void toConsole() {
//     String2.toConsole("Conjunto de Entrada: %s\nSuma objetivo: %d", numeros, SUMA);
// }

// Test de la lectura del fichero
public static void main(String[] args) {
    iniDatos("ficheros/Ejercicio2DatosEntrada1.txt");
//    System.out.println(getListaCentros());
}
}

```


SolucionEjercicio2.java

```

package _soluciones;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.stream.Collectors;

import _datos.DatosEjercicio2;
import _datos.DatosEjercicio2.Curso;
import us.lsi.common.List2;

public class SolucionEjercicio2 {

    public static SolucionEjercicio2 of_Range(List<Integer> ls) {
        return new SolucionEjercicio2(ls);
    }

    private double costoTotal;
    private List<Curso> cursos;

    private SolucionEjercicio2() {
        costoTotal = 0.;
        cursos = List2.empty();
    }

    private SolucionEjercicio2(List<Integer> ls) {
        costoTotal = 0;
        cursos = new ArrayList<>();

        for (int i = 0; i < ls.size(); i++) {
            if (ls.get(i) > 0) {
                Curso curso = DatosEjercicio2.getListasCursos().get(i);
                costoTotal += curso.coste();
                cursos.add(curso);
            }
        }
    }

    public static SolucionEjercicio2 empty() {
        return new SolucionEjercicio2();
    }

    public static SolucionEjercicio2 create(List<Integer> ls) {
        return new SolucionEjercicio2(ls);
    }

    @Override
    public String toString() {
        String s = cursos.stream().map(e -> "S" + e.id())
            .collect(Collectors.joining(", ", "Cursos elegidos: {", "}\n"));
        return String.format("%sCoste Total: %.1f", s, costoTotal);
    }
}

```

Ejercicio2AG.java

```

package ejercicio2;

import java.util.ArrayList;
import java.util.List;

import _datos.DatosEjercicio2;
import _soluciones.SolucionEjercicio2;
import us.lsi.ag.ValuesInRangeData;
import us.lsi.ag.agchromosomes.ChromosomeFactory.ChromosomeType;
import us.lsi.common.List2;

public class Ejercicio2AG implements ValuesInRangeData<Integer, SolucionEjercicio2> {

    public Ejercicio2AG(String linea) {
        DatosEjercicio2.iniDatos(linea);
    }

    @Override
    public Integer size() {
        // TODO Auto-generated method stub
        return DatosEjercicio2.getCursos();
    }

    @Override
    public ChromosomeType type() {
        // TODO Auto-generated method stub
        return ChromosomeType.Range;
    }

    @Override
    public Double fitnessFunction(List<Integer> value) {
        // TODO Auto-generated method stub
        Double goal = 0.;
        for (int i=0; i<value.size(); i++) {
            goal += value.get(i) * DatosEjercicio2.cursos.get(i).coste();
        }

        //Fitness Maximo
        Double fM = 0.;
        for (int i=0; i<value.size(); i++) {
            fM += DatosEjercicio2.cursos.get(i).coste() *
DatosEjercicio2.maximoCoste()*2;
        }

        //Restricciones
        Integer restriccion1 = 0;
        Integer restriccion2 = 0;

        List<Integer> tematicas = List2.ofTam(0, DatosEjercicio2.getTematicas());
        System.out.println(tematicas);
        for (int j = 0; j < DatosEjercicio2.getTematicas(); j++) {
            for (int i = 0; i < value.size(); i++) {
                if(DatosEjercicio2.seleccionaTematica(i, j) * value.get(i) == 1) {
                    tematicas.set(j, 1);
                    break;
                }
            }
        }
        /*

```

```

    * Recorro las tematicas
    * Dentro de una tematica, compruebo todos los cursos
    * si se selecciona el curso y ese curso tiene esa tematica, añado 1
    * si toda la lista esta rellena de 1, entonces se cubren todas las
tematicas
    */
}

restriccion1 = tematicas.stream().allMatch(x->x == 1)?0:1;

Integer maxCentros = DatosEjercicio2.getCentrosDiferentes();
List<Integer> centros = new ArrayList<>();
for (int i = 0; i < value.size(); i++) {
    if(value.get(i) == 1) {
        centros.add(DatosEjercicio2.getListaCursos().get(i).centro());
    }
    /*
    * Recorro los cursos y almaceno los centros en una lista
    * compruebo cuantos centros diferentes hay y devuelvo 1 si
    * hay más de maxCentros y 0 en caso contrario.
    */
}
restriccion2 = centros.stream().distinct().count()<=maxCentros?0:1;

return 0.0 - goal - (restriccion1 * fM) - (restriccion2 * fM);
}

@Override
public SolucionEjercicio2 solucion(List<Integer> value) {
    System.out.println(value); //esto es solo para hacer pruebas

    return SolucionEjercicio2.create(value);
}

@Override
public Integer max(Integer i) {
    // TODO Auto-generated method stub
    return 2;
}

@Override
public Integer min(Integer i) {
    // TODO Auto-generated method stub
    return 0;
}

}

```

Ejercicio2PLE.java

```

package ejercicio2;

import java.io.IOException;
import java.util.Locale;

import _datos.DatosEjercicio2;
import us.lsi.common.String2;
import us.lsi.gurobi.GurobiLp;
import us.lsi.gurobi.GurobiSolution;
import us.lsi.solve.AuxGrammar;

public class Ejercicio2PLE {

    public static void ejercicio2_model() throws IOException {
        //Leer Datos de entrada
        DatosEjercicio2.iniDatos("ficheros/Ejercicio2DatosEntrada1.txt");

        //si cambia el fichero de datos de entrada, cambiar tambien el nº del .lp para no
        //sobreescribirlo
        //Pasar de LSI a Gurobi
        AuxGrammar.generate                                     //generar un archivo gurobi
            (DatosEjercicio2.class,                             //la clase que utilizo
            "lsi_models/ejercicio2.lsi",                         //de donde saco el modelo
            "gurobi_models/Ejercicio2-1.lp"); //donde lo llevo

        //Usar Gurobi
        GurobiSolution solution = GurobiLp.gurobi("gurobi_models/Ejercicio2-1.lp");
        //Interpretar solucion
        //String2.toConsole("\nSolucion PLE: %s", solution.toString((s,d)-
        >d>0.).substring(2));

        Locale.setDefault(new Locale("en", "US"));
        System.out.println(solution.toString((s,d)->d>0.));
    }

    public static void main(String[] args) throws IOException {
        ejercicio2_model();
    }
}

```

Ejercicio2.lsi

head section

```
Integer getCursos()
Integer getTematicas()
Integer getCentros()
Integer getCentrosDiferentes()

Double getPrecioInscripcion(Integer i)
Integer seleccionaTematica(Integer i, Integer j)
Integer seleccionaCentro(Integer i, Integer k)
```

```
Integer n = getCursos()
Integer m = getTematicas()
Integer nc = getCentros()
Integer maxCentros = getCentrosDiferentes()
```

goal section

```
min sum(getPrecioInscripcion(i) x[i] , i in 0 .. n)
```

constraints section

```
sum(seleccionaTematica(i,j) x[i], i in 0 .. n) >= 1, j in 0 .. m
```

```
sum(y[k], k in 0 .. nc) <= maxCentros
```

```
seleccionaCentro(i,k) x[i] - y[k] <= 0, i in 0 .. n, k in 0 .. nc
```

bin

```
x[i], i in 0 .. n //se selecciona el curso i
y[k], k in 0 .. nc //se selecciona algun curso del centro k
```

TestEj2AGRange.java

```

package ejercicio2;

import java.util.List;
import java.util.Locale;

import _soluciones.SolucionEjercicio2;
import us.lsi.ag.agchromosomes.AlgoritmoAG;
import us.lsi.ag.agstopping.StoppingConditionFactory;

public class TestEj2AGRange {

    public static void main(String[] args) {
        Locale.setDefault(new Locale("en", "US"));

        AlgoritmoAG.ELITISM_RATE = 0.10;
        AlgoritmoAG.CROSSOVER_RATE = 0.95;
        AlgoritmoAG.MUTATION_RATE = 0.8;
        AlgoritmoAG.POPULATION_SIZE = 1000;

        StoppingConditionFactory.NUM_GENERATIONS = 1000;
        StoppingConditionFactory.stoppingConditionType =
        StoppingConditionFactory.StoppingConditionType.GenerationCount;

        for (int i = 1; i < 4; i++) {
            Ejercicio2AG p = new Ejercicio2AG("ficheros/Ejercicio2DatosEntrada" + i +
            ".txt");

            AlgoritmoAG<List<Integer>, SolucionEjercicio2> ap = AlgoritmoAG.of(p);
            ap.ejecuta();
            System.out.println("=====");
            System.out.println(ap.bestSolution());
            System.out.println("=====\\n");
        }
    }
}

```

Resultados AG

```
[Curso[id=0, tematicas=[1, 2, 3, 4], coste=10.0, centro=0], Curso[id=1, tematicas=[1, 4], coste=3.0, centro=0], Curso[id=2, tematicas=[5], coste=1.5, centro=1], ^
Curso[id=3, tematicas=[5], coste=5.0, centro=0]]
Max_Centros = 1
=====
[1, 0, 0, 1]
Cursos elegidos: {S0, S3}
Coste Total: 15.0
=====

[Curso[id=4, tematicas=[2, 3], coste=2.0, centro=0], Curso[id=5, tematicas=[4], coste=3.0, centro=0], Curso[id=6, tematicas=[1, 5], coste=5.0, centro=0], Curso
[id=7, tematicas=[1, 3, 4], coste=3.5, centro=2], Curso[id=8, tematicas=[4, 5], coste=1.5, centro=1]]
Max_Centros = 2
=====
[1, 0, 1, 0, 1]
Cursos elegidos: {S4, S6, S8}
Coste Total: 8.5
=====

[Curso[id=9, tematicas=[2, 6, 7], coste=2.0, centro=2], Curso[id=10, tematicas=[7], coste=3.0, centro=0], Curso[id=11, tematicas=[1, 5], coste=5.0, centro=0],
Curso[id=12, tematicas=[1, 3, 4], coste=3.5, centro=2], Curso[id=13, tematicas=[3, 7], coste=1.5, centro=1], Curso[id=14, tematicas=[4, 5, 6], coste=4.5,
centro=0], Curso[id=15, tematicas=[5, 6], coste=6.0, centro=1], Curso[id=16, tematicas=[2, 3, 5], coste=1.0, centro=1]]
Max_Centros = 3]
=====
[1, 0, 0, 1, 0, 0, 1]
Cursos elegidos: {S9, S12, S16}
Coste Total: 6.5
=====
```

Resultados PLE

Minimize

$$10.0 \ x_0 + 3.0 \ x_1 + 1.5 \ x_2 + 5.0 \ x_3$$

Subject To

$$a0: 1 \ x_0 + 1 \ x_1 + 0 \ x_2 + 0 \ x_3 \geq 1$$

$$a1: 1 \ x_0 + 0 \ x_1 + 0 \ x_2 + 0 \ x_3 \geq 1$$

$$a2: 1 \ x_0 + 0 \ x_1 + 0 \ x_2 + 0 \ x_3 \geq 1$$

$$a3: 1 \ x_0 + 1 \ x_1 + 0 \ x_2 + 0 \ x_3 \geq 1$$

$$a4: 0 \ x_0 + 0 \ x_1 + 1 \ x_2 + 1 \ x_3 \geq 1$$

$$b0: y_0 + y_1 \leq 1$$

$$c0: 1 \ x_0 - y_0 \leq 0$$

$$c1: 0 \ x_0 - y_1 \leq 0$$

$$c2: 1 \ x_1 - y_0 \leq 0$$

$$c3: 0 \ x_1 - y_1 \leq 0$$

$$c4: 0 \ x_2 - y_0 \leq 0$$

$$c5: 1 \ x_2 - y_1 \leq 0$$

$$c6: 1 \ x_3 - y_0 \leq 0$$

$$c7: 0 \ x_3 - y_1 \leq 0$$

Binary

$$x_0 \ x_1 \ x_2 \ x_3 \ y_0 \ y_1$$

End

Minimize

$$2.0 \ x_0 + 3.0 \ x_1 + 5.0 \ x_2 + 3.5 \ x_3 + 1.5 \ x_4$$

Subject To

$$\begin{aligned} a0: & 1 \ x_0 + 0 \ x_1 + 0 \ x_2 + 0 \ x_3 + 0 \ x_4 \geq 1 \\ a1: & 1 \ x_0 + 0 \ x_1 + 0 \ x_2 + 1 \ x_3 + 0 \ x_4 \geq 1 \\ a2: & 0 \ x_0 + 1 \ x_1 + 0 \ x_2 + 1 \ x_3 + 1 \ x_4 \geq 1 \\ a3: & 0 \ x_0 + 0 \ x_1 + 1 \ x_2 + 1 \ x_3 + 0 \ x_4 \geq 1 \\ a4: & 0 \ x_0 + 0 \ x_1 + 1 \ x_2 + 0 \ x_3 + 1 \ x_4 \geq 1 \\ b0: & y_0 + y_1 + y_2 \leq 2 \\ c0: & 1 \ x_0 - y_0 \leq 0 \\ c1: & 0 \ x_0 - y_1 \leq 0 \\ c2: & 0 \ x_0 - y_2 \leq 0 \\ c3: & 1 \ x_1 - y_0 \leq 0 \\ c4: & 0 \ x_1 - y_1 \leq 0 \\ c5: & 0 \ x_1 - y_2 \leq 0 \\ c6: & 1 \ x_2 - y_0 \leq 0 \\ c7: & 0 \ x_2 - y_1 \leq 0 \\ c8: & 0 \ x_2 - y_2 \leq 0 \\ c9: & 0 \ x_3 - y_0 \leq 0 \\ c10: & 1 \ x_3 - y_1 \leq 0 \\ c11: & 0 \ x_3 - y_2 \leq 0 \\ c12: & 0 \ x_4 - y_0 \leq 0 \\ c13: & 0 \ x_4 - y_1 \leq 0 \\ c14: & 1 \ x_4 - y_2 \leq 0 \end{aligned}$$

Binary

$$x_0 \ x_1 \ x_2 \ x_3 \ x_4 \ y_0 \ y_1 \ y_2$$

End

Minimize

$$2.0 \ x_0 + 3.0 \ x_1 + 5.0 \ x_2 + 3.5 \ x_3 + 1.5 \ x_4 + 4.5 \ x_5 + 6.0 \ x_6 + 1.0 \ x_7$$

Subject To

$$\begin{aligned} a0: & 1 \ x_0 + 0 \ x_1 + 0 \ x_2 + 0 \ x_3 + 0 \ x_4 + 0 \ x_5 + 0 \ x_6 + 1 \ x_7 \geq 1 \\ a1: & 1 \ x_0 + 0 \ x_1 + 0 \ x_2 + 0 \ x_3 + 0 \ x_4 + 1 \ x_5 + 1 \ x_6 + 0 \ x_7 \geq 1 \\ a2: & 1 \ x_0 + 1 \ x_1 + 0 \ x_2 + 0 \ x_3 + 1 \ x_4 + 0 \ x_5 + 0 \ x_6 + 0 \ x_7 \geq 1 \\ a3: & 0 \ x_0 + 0 \ x_1 + 1 \ x_2 + 1 \ x_3 + 0 \ x_4 + 0 \ x_5 + 0 \ x_6 + 0 \ x_7 \geq 1 \\ a4: & 0 \ x_0 + 0 \ x_1 + 1 \ x_2 + 0 \ x_3 + 0 \ x_4 + 1 \ x_5 + 1 \ x_6 + 1 \ x_7 \geq 1 \\ a5: & 0 \ x_0 + 0 \ x_1 + 0 \ x_2 + 1 \ x_3 + 1 \ x_4 + 0 \ x_5 + 0 \ x_6 + 1 \ x_7 \geq 1 \\ a6: & 0 \ x_0 + 0 \ x_1 + 0 \ x_2 + 1 \ x_3 + 0 \ x_4 + 1 \ x_5 + 0 \ x_6 + 0 \ x_7 \geq 1 \\ b0: & y_0 + y_1 + y_2 \leq 3 \\ c0: & 1 \ x_0 - y_0 \leq 0 \\ c1: & 0 \ x_0 - y_1 \leq 0 \\ c2: & 0 \ x_0 - y_2 \leq 0 \\ c3: & 0 \ x_1 - y_0 \leq 0 \\ c4: & 1 \ x_1 - y_1 \leq 0 \\ c5: & 0 \ x_1 - y_2 \leq 0 \\ c6: & 0 \ x_2 - y_0 \leq 0 \\ c7: & 1 \ x_2 - y_1 \leq 0 \\ c8: & 0 \ x_2 - y_2 \leq 0 \\ c9: & 1 \ x_3 - y_0 \leq 0 \\ c10: & 0 \ x_3 - y_1 \leq 0 \\ c11: & 0 \ x_3 - y_2 \leq 0 \\ c12: & 0 \ x_4 - y_0 \leq 0 \\ c13: & 0 \ x_4 - y_1 \leq 0 \\ c14: & 1 \ x_4 - y_2 \leq 0 \\ c15: & 0 \ x_5 - y_0 \leq 0 \\ c16: & 1 \ x_5 - y_1 \leq 0 \\ c17: & 0 \ x_5 - y_2 \leq 0 \\ c18: & 0 \ x_6 - y_0 \leq 0 \\ c19: & 0 \ x_6 - y_1 \leq 0 \\ c20: & 1 \ x_6 - y_2 \leq 0 \\ c21: & 0 \ x_7 - y_0 \leq 0 \\ c22: & 0 \ x_7 - y_1 \leq 0 \\ c23: & 1 \ x_7 - y_2 \leq 0 \end{aligned}$$

Binary

$$x_0 \ x_1 \ x_2 \ x_3 \ x_4 \ x_5 \ x_6 \ x_7 \ y_0 \ y_1 \ y_2$$

End

Ejercicio3

DatosEjercicio3.java

```
package _datos;

import java.util.Arrays;
import java.util.Comparator;
import java.util.List;
import java.util.Map;
import java.util.Set;
import java.util.stream.Collectors;

import us.lsi.common.Files2;

public class DatosEjercicio3 {

    public static List<Investigador> investigadores;
    public static record Investigador(String nombre, Integer capacidad, Integer
especialidad) {
        public Investigador of(String nombre, Integer capacidad, Integer especialidad) {
            return new Investigador(nombre, capacidad, especialidad);
        }

        public static Investigador ofFormat(String linea) {
            String[] formato = linea.split(":");
            String nombre = formato[0];
            String segundaParte = formato[1].trim();
            String[] segundaPartePars = segundaParte.split(";");
            Integer capacidad =
Integer.parseInt(segundaPartePars[0].split("=")[1].trim());
            Integer especialidad =
Integer.parseInt(segundaPartePars[1].split("=")[1].trim());
            return new Investigador(nombre, capacidad, especialidad);
        }
    }

    public static List<Trabajo> trabajos;
    public static record Trabajo(String nombre, Integer calidad, Map<Integer, Integer>
reparto) {
        public Trabajo of(String nombre, Integer calidad, Map<Integer, Integer> reparto) {
            return new Trabajo(nombre, calidad, reparto);
        }

        public static Trabajo ofFormat(String linea) {
            String[] formato = linea.split("->");
            String nombre = formato[0].trim();
            String segundaParte = formato[1].trim();
            String[] segundaPartePars = segundaParte.split(";");
            Integer calidad =
Integer.parseInt(segundaPartePars[0].split("=")[1].trim());
            Map<Integer, Integer> reparto =
Arrays.stream(segundaPartePars[1].split("=")[1].split(","))
                .map(pair -> pair.split(":"))
                .collect(Collectors.toMap(
                    keyValue -> Integer.parseInt(keyValue[0].replace("(",
"".replace(")", "")),
                    keyValue -> Integer.parseInt(keyValue[1].replace("(",
"".replace(")", ""))
                ));
        }
    }
}
```

```

        return new Trabajo(nombre, calidad, reparto);
    }
}

public static void iniDatos(String fichero) {
    List<String> lineas = Files2.LinesFromFile(fichero);
    investigadores = lineas.stream().filter(l -> l.startsWith("I")).map(x ->
Investigador.ofFormat(x)).toList();
    trabajos = lineas.stream().filter(l -> l.startsWith("T")).map(x ->
Trabajo.ofFormat(x)).toList();
    toConsole();
}

//.....
//head section del .lsi

public static Integer getTrabajos() {
    return trabajos.size();
}

public static Integer getInvestigadores() {
    return investigadores.size();
}

public static Integer getCapacidad(Integer i) {
    return investigadores.get(i).capacidad;
}
public static Integer getCalidad(Integer i) {
    return trabajos.get(i).calidad;
}

public static Integer getMaximoDias() {
    return investigadores.stream().map(Investigador::capacidad).reduce((a, b) ->
a>b?a:b).get();
}

// public static Integer compruebaTrabajo(Integer i, Integer k) {
//     Integer checkInv = investigadores.get(i).especialidad();
//     Set<Integer> checkTra = trabajos.get(k).reparto().keySet();
//     return checkTra.contains(checkInv)?1:0;
// }

public static Integer diasNecesito(Integer j, Integer k) {
    return trabajos.get(j).reparto().get(k);
}

public static Integer getEspecialidades() {
    return
    investigadores.stream().map(Investigador::especialidad).distinct().collect(Collectors.toList())
    .size();
}
public static Integer totalTrabajo(Integer j) {
    return
    trabajos.get(j).reparto().values().stream().mapToInt(Integer::intValue).sum();
}

public static Integer seleccionaEspecialidad(Integer i, Integer k) {
    return investigadores.get(i).especialidad().equals(k)?1:0;
}
//.....

```

```
public static List<Investigador> getListaInvestigadores(){
    return investigadores;
}
public static List<Trabajo> getListaTrabajos(){
    return trabajos;
}

public static void toConsole() {
    //String2.toConsole("Conjunto de Entrada Tipos: %s\nConjunto de Entrada
Variedades: %d", tipos, variedades);
    System.out.println(investigadores);
    System.out.println(trabajos);
}

// Test de la lectura del fichero
public static void main(String[] args) {
    iniDatos("ficheros/Ejercicio3DatosEntrada1.txt");
}
}
```

SolucionEjercicio3.java

```

package _soluciones;

import java.util.List;
import java.util.stream.Collectors;

import _datos.DatosEjercicio3;
import _datos.DatosEjercicio3.Investigador;
import us.lsi.common.List2;

public class SolucionEjercicio3 {

    public static SolucionEjercicio3 of_Range(List<Integer> ls) {
        return new SolucionEjercicio3(ls);
    }

    private Integer calidad;
    private List<Investigador> investigadores;
    private List<List<Integer>> solucion;

    //List(0,...,n-1, n,...2n-1, ...)
    //    i0,...,in, i0,...,in, ...
    //      j0          j1      ...

    private SolucionEjercicio3() {
        calidad = 0;
        investigadores = List2.empty();
        solucion = List2.empty();
    }

    private SolucionEjercicio3(List<Integer> ls) {
        //variables aux
        Integer numTrabajos = DatosEjercicio3.getTrabajos();
        Integer numInvestigadores = DatosEjercicio3.getInvestigadores();
        Integer numEspecialidades = DatosEjercicio3.getEspecialidades();

        calidad = 0;
        investigadores = DatosEjercicio3.investigadores;
        solucion = List2.empty();
        //para hacer cada i0,...,in, i0,...,in, ...
        for (int i=0; i<numInvestigadores; i++) {
            solucion.add(List2.empty());
        }

        for (int j=0; j<numTrabajos; j++) {
            Integer trabajosValue = j * numInvestigadores;
            List<Integer> trabajoActual = ls.subList(trabajosValue,
trabajosValue+numInvestigadores); //me quedo con el j que quiero
            //actualizo las horas que cada trabajador i le dedica al trabajo j
            for (int i=0; i<numInvestigadores; i++) {
                solucion.get(i).add(trabajoActual.get(i));
            }
            Boolean trabaja=true;
            for (int k=0; k<numEspecialidades; k++) {
                Integer suma=0;
                for (int i=0; i<numInvestigadores; i++) {
                    suma +=
trabajoActual.get(i)*DatosEjercicio3.seleccionaEspecialidad(i, k); //sum(x[i,j], i in 0 .. n |
seleccionaEspecialidad(i, k) = 1)
                }
            }
        }
    }

```

```

        if (suma < DatosEjercicio3.diasNecesito(j, k)) {
            trabaja = false; //y[j] = 0
            k = numEspecialidades;
        }
    }

    if (trabaja) { //y[j] = 1
        calidad += DatosEjercicio3.getCalidad(j);
    }
}

public static SolucionEjercicio3 empty() {
    return new SolucionEjercicio3();
}

@Override
public String toString() {
    String s = investigadores.stream()
        .map(i -> "INVESTIGADOR " + i.nombre() + ": " + i)
        .collect(Collectors.joining("\n", "Reparto de horas:\n", "\n"));
    return String.format("%sSuma de las calidades de los trabajos realizados: %d", s,
calidad);
}
}

```

Ejercicio3AG.java

```

package ejercicio3;

import java.util.List;

import _datos.DatosEjercicio2;
import _datos.DatosEjercicio3;
import _datos.DatosEjercicio3.Investigador;
import _soluciones.SolucionEjercicio3;
import us.lsi.ag.ValuesInRangeData;
import us.lsi.ag.agchromosomes.ChromosomeFactory.ChromosomeType;

public class Ejercicio3AG implements ValuesInRangeData<Integer, SolucionEjercicio3> {

    public Ejercicio3AG(String linea) {
        DatosEjercicio3.iniDatos(linea);
    }

    @Override
    public Integer size() {
        // TODO Auto-generated method stub
        return DatosEjercicio3.getTrabajos() * DatosEjercicio3.getInvestigadores();
    }
    //List(0,...,n-1, n,...2n-1, ...)

    //      i0,...,in,  i0,...,in,  ...
    //      j0          j1          ...
    // para acceder a cada j hago la division
    // para acceder a cada i hago el módulo

}

@Override
public ChromosomeType type() {
    // TODO Auto-generated method stub
    return ChromosomeType.Range;
}

@Override
public Double fitnessFunction(List<Integer> value) {
    // TODO Auto-generated method stub

    //variables aux
    Integer numTrabajos = DatosEjercicio3.getTrabajos();
    Integer numInvestigadores = DatosEjercicio3.getInvestigadores();
    Integer numEspecialidades = DatosEjercicio3.getEspecialidades();
    Integer capacidadUso = 0;
    Integer restriccion1 = 0;
    Integer restriccion2 = 0;

    //      max sum(getCalidad(j) y[j], j in 0 .. m)
    Double goal = 0.;

    //Fitness Maximo
    Double fM = 0.;
    Double fmAux = 0.;
    for (int i = 0; i < numTrabajos; i++) {
        fmAux+= DatosEjercicio3.getCalidad(i);
    }
}

```



```

    }

    fM = Math.pow(fmAux, 2);

    //sum(x[i,j], j in 0 .. m) <= getCapacidad(i), i in 0 .. n

    for (int i=0; i<numInvestigadores; i++) {
        capacidadUso=0; //comienzo dedicando 0 horas
        for (int invValue=i; invValue<value.size(); invValue+=numInvestigadores) {
//Itero en la lista n*m el investigador i en cada uno de los trabajos
            capacidadUso += value.get(invValue);
            //voy actualizando las horas que le dedica este investigador i
        }
        Investigador investigadorActual = DatosEjercicio3.investigadores.get(i);
        if (capacidadUso > investigadorActual.capacidad()) restriccion1 +=
capacidadUso-investigadorActual.capacidad(); //voy sumando lo lejos que me quedo
    }

    //sum(x[i,j], i in 0 .. n | seleccionaEspecialidad(i, k) = 1) - diasNecesito(j, k)
    y[j] = 0, j in 0 .. m, k in 0 .. e
    /*
    * for(int j=0...)
    *     for(int k = 0...)
    *         for(int i = 0...)
    *             diasUso = sum(x[i,j], i in 0 .. n |
seleccionaEspecialidad(i, k) = 1)
    *             diasUso - diasNecesito(j, k) y[j] = 0
    */

    for (int j=0; j<numTrabajos; j++) {
        Integer trabajosValue = j*numInvestigadores;//j0,j1,...,jm en values
        List<Integer> trab = value.subList(trabajosValue,
trabajosValue+numInvestigadores);
        Boolean trabaja=true;
        for (int k=0; k<numEspecialidades; k++) {
            Integer diasUso=0;
            for (int i=0; i<numInvestigadores; i++) { //sum(x[i,j], i in 0 .. n |
seleccionaEspecialidad(i, k) = 1)
                diasUso +=
trab.get(i)*DatosEjercicio3.seleccionaEspecialidad(i, k);
            }

            if (diasUso != DatosEjercicio3.diasNecesito(j, k)) {
                trabaja = false; //y[j] = 0
                restriccion2 += Math.abs(diasUso -
DatosEjercicio3.diasNecesito(j, k)); //voy sumando lo lejos que me quedo
            }
        }
        if (trabaja) { //y[j] = 1
            goal += DatosEjercicio3.getCalidad(j);
        }
    }

    return 0.0 - goal - (restriccion1 * fM) - (restriccion2 * fM) ;
}

@Override
public SolucionEjercicio3 solucion(List<Integer> value) {

```

```
        System.out.println(value); //esto es solo para hacer pruebas
        return SolucionEjercicio3.of_Range(value);
    }

    @Override
    public Integer max(Integer i) {
        // TODO Auto-generated method stub
        Integer i_aux = i%DatosEjercicio3.getInvestigadores(); //para cada trabajo j, veo
el investigador i que más horas dedica
        return DatosEjercicio3.getCapacidad(i_aux)+1;
    }

    @Override
    public Integer min(Integer i) {
        // TODO Auto-generated method stub
        return 0;
    }
}
```

Ejercicio3PLE.java

```

package ejercicio3;

import java.io.IOException;
import java.util.List;

import _datos.DatosEjercicio2;
import _datos.DatosEjercicio3;
import _datos.DatosEjercicio3.Investigador;
import _datos.DatosEjercicio3.Trabajo;
import us.lsi.common.String2;
import us.lsi.gurobi.GurobiLp;
import us.lsi.gurobi.GurobiSolution;
import us.lsi.solve.AuxGrammar;

public class Ejercicio3PLE {

    public static void ejercicio3_model() throws IOException {
        //Leer Datos de entrada
        DatosEjercicio3.iniDatos("ficheros/Ejercicio3DatosEntrada1.txt");

        //si cambia el fichero de datos de entrada, cambiar tambien el nº del .lp para no
        //sobreescribirlo
        //Pasar de LSI a Gurobi
        AuxGrammar.generate
            (DatosEjercicio3.class,
            "lsi_models/ejercicio3.lsi",
            "gurobi_models/Ejercicio3-1.lp"); //generar un archivo gurobi
            //la clase que utilizo
            //de donde saco el modelo
            //donde lo llevo
        //Usar Gurobi
        GurobiSolution solution = GurobiLp.gurobi("gurobi_models/Ejercicio3-1.lp");
        //Interpretar solucion
        String2.toConsole("\nSolucion PLE: %s", solution.toString((s,d)-
        >d>0.).substring(2));

        //      Locale.setDefault(new Locale("en", "US"));
        //      System.out.println(solution.toString((s,d)->d>0.));
    }

    public static void main(String[] args) throws IOException {
        ejercicio3_model();
    }

}

```

Ejercicio3.lsi

head section

```
Integer getTrabajos()
Integer getInvestigadores()
Integer getEspecialidades()

Integer getCapacidad(Integer i)
Integer getCalidad(Integer i)
Integer diasNecesito(Integer j, Integer k)
Integer seleccionaEspecialidad(Integer j, Integer k)
Integer totalTrabajo(Integer j)
```

```
Integer n = getInvestigadores()
Integer m = getTrabajos()
Integer e = getEspecialidades()
```

goal section

```
max sum(getCalidad(j) y[j], j in 0 .. m)
```

constraints section

```
sum(x[i,j], j in 0 .. m) <= getCapacidad(i), i in 0 .. n
```

```
//tt(j) <= sum(x[i, j]) | ei(i) == et(j))
sum(x[i,j], i in 0 .. n | seleccionaEspecialidad(i, k) = 1) - diasNecesito(j, k) y[j] = 0, j in
0 .. m, k in 0 .. e
```

bounds section

```
x[i,j] <= getCapacidad(j) + 1, i in 0 .. n, j in 0 .. m
```

int

```
x[i,j], i in 0 .. n, j in 0 .. m
```

bin

```
y[j], j in 0 .. m
```

TestEj3AGRange.java

```

package ejercicio3;

import java.util.List;
import java.util.Locale;

import _soluciones.SolucionEjercicio3;
import _soluciones.SolucionEjercicio4;
import ejercicio4.Ejercicio4AG;
import us.lsi.ag.agchromosomes.AlgoritmoAG;
import us.lsi.ag.agstopping.StoppingConditionFactory;

public class TestEj3AGRange {

    public static void main(String[] args) {
        Locale.setDefault(new Locale("en", "US"));

        AlgoritmoAG.ELITISM_RATE = 0.10;
        AlgoritmoAG.CROSSOVER_RATE = 0.95;
        AlgoritmoAG.MUTATION_RATE = 0.8;
        AlgoritmoAG.POPULATION_SIZE = 1000;

        StoppingConditionFactory.NUM_GENERATIONS = 1000;
        StoppingConditionFactory.stoppingConditionType =
        StoppingConditionFactory.StoppingConditionType.GenerationCount;

        for (int i = 1; i < 4; i++) {
            Ejercicio3AG p = new Ejercicio3AG("ficheros/Ejercicio3DatosEntrada" + i +
            ".txt");

            AlgoritmoAG<List<Integer>, SolucionEjercicio3> ap = AlgoritmoAG.of(p);
            ap.ejecuta();
            System.out.println("=====");
            System.out.println(ap.bestSolution());
            System.out.println("=====\\n");
        }
    }
}

```

Resultados AG

```
[Investigador[nombre=INV1, capacidad=6, especialidad=0], Investigador[nombre=INV2, capacidad=3, especialidad=1], Investigador[nombre=INV3, capacidad=8, especialidad=2],
Trabajo[nombre=T01, calidad=5, reparto={0=6, 1=0, 2=0}], Trabajo[nombre=T02, calidad=10, reparto={0=0, 1=3, 2=8}]]
=====
[6, 0, 0, 0, 3, 0]
Reparto de horas:
INVESTIGADOR INV1: Investigador[nombre=INV1, capacidad=6, especialidad=0]
INVESTIGADOR INV2: Investigador[nombre=INV2, capacidad=3, especialidad=1]
INVESTIGADOR INV3: Investigador[nombre=INV3, capacidad=8, especialidad=2]
Suma de las calidades de los trabajos realizados: 15
=====

[Investigador[nombre=INV1, capacidad=10, especialidad=0], Investigador[nombre=INV2, capacidad=5, especialidad=1], Investigador[nombre=INV3, capacidad=8, especialidad=2],
Trabajo[nombre=T01, calidad=7, reparto={0=2, 1=0, 2=5, 3=0}], Trabajo[nombre=T02, calidad=9, reparto={0=8, 1=4, 2=3, 3=0}], Trabajo[nombre=T03, calidad=5, reparto={0=0, 1=3, 2=8}]]
=====
[2, 0, 5, 0, 0, 6, 4, 3, 2, 0, 2, 0, 0, 0, 5]
Reparto de horas:
INVESTIGADOR INV1: Investigador[nombre=INV1, capacidad=10, especialidad=0]
INVESTIGADOR INV2: Investigador[nombre=INV2, capacidad=5, especialidad=1]
INVESTIGADOR INV3: Investigador[nombre=INV3, capacidad=8, especialidad=2]
INVESTIGADOR INV4: Investigador[nombre=INV4, capacidad=2, especialidad=0]
INVESTIGADOR INV5: Investigador[nombre=INV5, capacidad=5, especialidad=3]
Suma de las calidades de los trabajos realizados: 16
=====

[Investigador[nombre=INV1, capacidad=1, especialidad=2], Investigador[nombre=INV2, capacidad=10, especialidad=1], Investigador[nombre=INV3, capacidad=3, especialidad=0],
Trabajo[nombre=T01, calidad=8, reparto={0=2, 1=0, 2=2, 3=0}], Trabajo[nombre=T02, calidad=5, reparto={0=8, 1=5, 2=4, 3=2}], Trabajo[nombre=T03, calidad=8, reparto={0=0, 1=3, 2=8}]]
=====
[1, 0, 0, 0, 0, 0, 1, 0, 0, 5, 3, 4, 0, 1, 1, 1, 0, 0, 0, 0, 3, 0, 0, 12, 0, 7, 0, 0, 1, 1, 1, 3, 0, 3, 2, 3, 1, 1, 0, 0]
Reparto de horas:
INVESTIGADOR INV1: Investigador[nombre=INV1, capacidad=1, especialidad=2]
INVESTIGADOR INV2: Investigador[nombre=INV2, capacidad=10, especialidad=1]
INVESTIGADOR INV3: Investigador[nombre=INV3, capacidad=3, especialidad=0]
INVESTIGADOR INV4: Investigador[nombre=INV4, capacidad=4, especialidad=0]
INVESTIGADOR INV5: Investigador[nombre=INV5, capacidad=10, especialidad=3]
INVESTIGADOR INV6: Investigador[nombre=INV6, capacidad=4, especialidad=3]
INVESTIGADOR INV7: Investigador[nombre=INV7, capacidad=1, especialidad=2]
INVESTIGADOR INV8: Investigador[nombre=INV8, capacidad=30, especialidad=3]
Suma de las calidades de los trabajos realizados: 0
```

Activar Windows

Resultados PLE

Maximize

$$5 y_0 + 10 y_1$$

Subject To

$$a0: x_{0_0} + x_{0_1} \leq 6$$

$$a1: x_{1_0} + x_{1_1} \leq 3$$

$$a2: x_{2_0} + x_{2_1} \leq 8$$

$$b0: x_{0_0} - 6 y_0 = 0$$

$$b1: x_{1_0} - 0 y_0 = 0$$

$$b2: x_{2_0} - 0 y_0 = 0$$

$$b3: x_{0_1} - 0 y_1 = 0$$

$$b4: x_{1_1} - 3 y_1 = 0$$

$$b5: x_{2_1} - 8 y_1 = 0$$

Bounds

$$x_{0_0} \leq 7$$

$$x_{0_1} \leq 4$$

$$x_{1_0} \leq 7$$

$$x_{1_1} \leq 4$$

$$x_{2_0} \leq 7$$

$$x_{2_1} \leq 4$$

General

$$x_{0_0} \quad x_{0_1} \quad x_{1_0} \quad x_{1_1} \quad x_{2_0} \quad x_{2_1}$$

End

Maximize

$$7 y_0 + 9 y_1 + 5 y_2$$

Subject To

$$\begin{aligned} a0: x_{0_0} + x_{0_1} + x_{0_2} &\leq 10 \\ a1: x_{1_0} + x_{1_1} + x_{1_2} &\leq 5 \\ a2: x_{2_0} + x_{2_1} + x_{2_2} &\leq 8 \\ a3: x_{3_0} + x_{3_1} + x_{3_2} &\leq 2 \\ a4: x_{4_0} + x_{4_1} + x_{4_2} &\leq 5 \\ b0: x_{0_0} + x_{3_0} - 2 y_0 &= 0 \\ b1: x_{1_0} - 0 y_0 &= 0 \\ b2: x_{2_0} - 5 y_0 &= 0 \\ b3: x_{4_0} - 0 y_0 &= 0 \\ b4: x_{0_1} + x_{3_1} - 8 y_1 &= 0 \\ b5: x_{1_1} - 4 y_1 &= 0 \\ b6: x_{2_1} - 3 y_1 &= 0 \\ b7: x_{4_1} - 0 y_1 &= 0 \\ b8: x_{0_2} + x_{3_2} - 2 y_2 &= 0 \\ b9: x_{1_2} - 0 y_2 &= 0 \\ b10: x_{2_2} - 0 y_2 &= 0 \\ b11: x_{4_2} - 7 y_2 &= 0 \end{aligned}$$

Bounds

$$\begin{aligned} x_{0_0} &\leq 11 \\ x_{0_1} &\leq 6 \\ x_{0_2} &\leq 9 \\ x_{1_0} &\leq 11 \\ x_{1_1} &\leq 6 \\ x_{1_2} &\leq 9 \\ x_{2_0} &\leq 11 \\ x_{2_1} &\leq 6 \\ x_{2_2} &\leq 9 \\ x_{3_0} &\leq 11 \\ x_{3_1} &\leq 6 \\ x_{3_2} &\leq 9 \\ x_{4_0} &\leq 11 \\ x_{4_1} &\leq 6 \\ x_{4_2} &\leq 9 \end{aligned}$$

General

$$x_{0_0} \ x_{0_1} \ x_{0_2} \ x_{1_0} \ x_{1_1} \ x_{1_2} \ x_{2_0} \ x_{2_1} \ x_{2_2} \ x_{3_0} \ x_{3_1} \ x_{3_2} \ x_{4_0} \ x_{4_1} \ x_{4_2}$$

End

Maximize

$$8 y_0 + 5 y_1 + 8 y_2 + 5 y_3 + 9 y_4$$

Subject To

$$\begin{aligned} a0: x_{0_0} + x_{0_1} + x_{0_2} + x_{0_3} + x_{0_4} &\leq 1 \\ a1: x_{1_0} + x_{1_1} + x_{1_2} + x_{1_3} + x_{1_4} &\leq 10 \\ a2: x_{2_0} + x_{2_1} + x_{2_2} + x_{2_3} + x_{2_4} &\leq 3 \\ a3: x_{3_0} + x_{3_1} + x_{3_2} + x_{3_3} + x_{3_4} &\leq 4 \\ a4: x_{4_0} + x_{4_1} + x_{4_2} + x_{4_3} + x_{4_4} &\leq 10 \\ a5: x_{5_0} + x_{5_1} + x_{5_2} + x_{5_3} + x_{5_4} &\leq 4 \\ a6: x_{6_0} + x_{6_1} + x_{6_2} + x_{6_3} + x_{6_4} &\leq 1 \\ a7: x_{7_0} + x_{7_1} + x_{7_2} + x_{7_3} + x_{7_4} &\leq 30 \\ b0: x_{2_0} + x_{3_0} - 2 y_0 &= 0 \\ b1: x_{1_0} - 0 y_0 &= 0 \\ b2: x_{0_0} + x_{6_0} - 2 y_0 &= 0 \\ b3: x_{4_0} + x_{5_0} + x_{7_0} - 0 y_0 &= 0 \\ b4: x_{2_1} + x_{3_1} - 8 y_1 &= 0 \\ b5: x_{1_1} - 5 y_1 &= 0 \\ b6: x_{0_1} + x_{6_1} - 4 y_1 &= 0 \\ b7: x_{4_1} + x_{5_1} + x_{7_1} - 2 y_1 &= 0 \\ b8: x_{2_2} + x_{3_2} - 0 y_2 &= 0 \\ b9: x_{1_2} - 5 y_2 &= 0 \\ b10: x_{0_2} + x_{6_2} - 0 y_2 &= 0 \\ b11: x_{4_2} + x_{5_2} + x_{7_2} - 15 y_2 &= 0 \\ b12: x_{2_3} + x_{3_3} - 0 y_3 &= 0 \\ b13: x_{1_3} - 7 y_3 &= 0 \\ b14: x_{0_3} + x_{6_3} - 8 y_3 &= 0 \\ b15: x_{4_3} + x_{5_3} + x_{7_3} - 5 y_3 &= 0 \\ b16: x_{2_4} + x_{3_4} - 5 y_4 &= 0 \\ b17: x_{1_4} - 5 y_4 &= 0 \\ b18: x_{0_4} + x_{6_4} - 0 y_4 &= 0 \\ b19: x_{4_4} + x_{5_4} + x_{7_4} - 2 y_4 &= 0 \end{aligned}$$

Bounds

$$\begin{aligned} x_{0_0} &\leq 2 \\ x_{0_1} &\leq 11 \\ x_{0_2} &\leq 4 \\ x_{0_3} &\leq 5 \\ x_{0_4} &\leq 11 \\ x_{1_0} &\leq 2 \\ x_{1_1} &\leq 11 \\ x_{1_2} &\leq 4 \\ x_{1_3} &\leq 5 \\ x_{1_4} &\leq 11 \\ x_{2_0} &\leq 2 \\ x_{2_1} &\leq 11 \\ x_{2_2} &\leq 4 \\ x_{2_3} &\leq 5 \\ x_{2_4} &\leq 11 \\ x_{3_0} &\leq 2 \\ x_{3_1} &\leq 11 \\ x_{3_2} &\leq 4 \\ x_{3_3} &\leq 5 \\ x_{3_4} &\leq 11 \end{aligned}$$

```

x_4_0 <= 2
x_4_1 <= 11
x_4_2 <= 4
x_4_3 <= 5
x_4_4 <= 11
x_5_0 <= 2
x_5_1 <= 11
x_5_2 <= 4
x_5_3 <= 5
x_5_4 <= 11
x_6_0 <= 2
x_6_1 <= 11
x_6_2 <= 4
x_6_3 <= 5
x_6_4 <= 11
x_7_0 <= 2
x_7_1 <= 11
x_7_2 <= 4
x_7_3 <= 5
x_7_4 <= 11

```

General

```

x_0_0 x_0_1 x_0_2 x_0_3 x_0_4 x_1_0 x_1_1 x_1_2 x_1_3 x_1_4 x_2_0 x_2_1 x_2_2 x_2_3 x_2_4
x_3_0 x_3_1 x_3_2 x_3_3 x_3_4 x_4_0 x_4_1 x_4_2 x_4_3 x_4_4 x_5_0 x_5_1 x_5_2 x_5_3 x_5_4 x_6_0
x_6_1 x_6_2 x_6_3 x_6_4 x_7_0 x_7_1 x_7_2 x_7_3 x_7_4

```

End

Ejercicio4

DatosEjercicio4.java

```
package _datos;

import java.util.ArrayList;
import java.util.List;

import org.jgrapht.Graph;

import us.lsi.graphs.Graphs2;
import us.lsi.graphs.GraphsReader;

public class DatosEjercicio4 {

    private static int id_aux = 0;
    public record Conexion(int id, Double distancia) {
        public static Conexion ofFormat(String[] formato) {
            Integer id = id_aux++;
            Double dist = Double.valueOf(formato[2].trim());
            return new Conexion(id, dist);
        }
    }

    public record Cliente(int id, Double beneficio) {
        public static Cliente ofFormat(String[] formato) {
            Integer id = Integer.valueOf(formato[0].trim());
            Double benef = Double.valueOf(formato[1].trim());
            return new Cliente(id, benef);
        }
    }

    public static Graph<Cliente, Conexion> g;
    public static void iniDatos(String fichero) {
        g = GraphsReader.newGraph(fichero, Cliente::ofFormat, Conexion::ofFormat,
            Graphs2::simpleWeightedGraph);
        toConsole();
    }

    public static Integer getNumVertices() {
        return g.vertexSet().size();
    }

    public static Cliente getCliente(Integer i) { //no puedo acceder a los elementos de un
    set con i porque no están ordenados, me hace falta un id
        List<Cliente> vertices = new ArrayList<>(g.vertexSet());
        return vertices.stream().filter(x -> x.id()==i).findFirst().get();
    }

    public static Double getBeneficio(Integer i) {
        return getCliente(i).beneficio();
    }

    public static Boolean existeArista(Integer i, Integer j) {
        Cliente c1 = getCliente(i);
        Cliente c2 = getCliente(j);
        return g.containsEdge(c1, c2);
    }

    public static Double getDistancia(Integer i, Integer j) {
        Cliente c1 = getCliente(i);
        Cliente c2 = getCliente(j);
        return g.getEdge(c1, c2).distancia();
    }
}
```

```
}  
private static void toConsole() {  
    System.out.println(g.vertexSet());  
    System.out.println(g.edgeSet());  
}  
public static void main(String[] args) {  
    iniDatos("ficheros/Ejercicio4DatosEntrada1.txt");  
    System.out.println(getDistancia(2, 4));  
}  
}
```

SolucionEjercicio4.java

```

package _soluciones;

import java.util.ArrayList;
import java.util.List;

import _datos.DatosEjercicio4;
import _datos.DatosEjercicio4.Cliente;

public class SolucionEjercicio4 {

    public static SolucionEjercicio4 of_Range(List<Integer> ls) {
        return new SolucionEjercicio4(ls);
    }

    private Double kms;
    private Double benef;
    private List<Cliente> clientes;

    private SolucionEjercicio4() {
        kms = 0.;
        benef = 0.;
        clientes = new ArrayList<>();
        Cliente c0 = DatosEjercicio4.getCliente(0);
        clientes.add(c0);
    }

    private SolucionEjercicio4(List<Integer> ls) {
        kms = 0.;
        benef = 0.;
        clientes = new ArrayList<>();
        Cliente c0 = DatosEjercicio4.getCliente(0);
        clientes.add(c0);
        for (int i = 0; i < ls.size(); i++) {
            Cliente c = DatosEjercicio4.getCliente(ls.get(i));
            clientes.add(c);
            if (i == 0) {
                if (DatosEjercicio4.existeArista(0, ls.get(i))) {
                    kms += DatosEjercicio4.getDistancia(0, ls.get(i));
                    benef += DatosEjercicio4.getBeneficio(ls.get(i)) - kms;
                }
            } else {
                if (DatosEjercicio4.existeArista(ls.get(i - 1), ls.get(i))) {
                    kms += DatosEjercicio4.getDistancia(ls.get(i - 1), ls.get(i));
                    benef += DatosEjercicio4.getBeneficio(ls.get(i)) - kms;
                }
            }
        }
    }

    public static SolucionEjercicio4 empty() {
        return new SolucionEjercicio4();
    }

    @Override
    public String toString() {
        List<Integer> ids = clientes.stream().map(c -> c.id()).toList();
        return "Camino a seguir:\n" + ids + "\nDistancia: " + kms + "\nBeneficio: " +
benef;
    }
}

```

}

Ejercicio4AG.java

```

package ejercicio4;

import java.util.List;

import _datos.DatosEjercicio4;
import _soluciones.SolucionEjercicio4;
import us.lsi.ag.SeqNormalData;
import us.lsi.ag.agchromosomes.ChromosomeFactory.ChromosomeType;

public class Ejercicio4AG implements SeqNormalData<SolucionEjercicio4> {

    public Ejercicio4AG(String linea) {
        DatosEjercicio4.iniDatos(linea);
    }

    @Override
    public ChromosomeType type() {
        // TODO Auto-generated method stub
        return ChromosomeType.Permutation;
    }

    @Override
    public Double fitnessFunction(List<Integer> value) {
        // TODO Auto-generated method stub
        //variables aux
        double goal = 0;
        double restriccion = 0;
        double fM = 0;
        double fmAux = 0;

        for (int i = 0; i < value.size(); i++) {
            if (i == 0) { //El reparto comienza desde una localización concreta (el
almacén)
                if (DatosEjercicio4.existeArista(0, value.get(i))) {
                    fmAux += DatosEjercicio4.getDistancia(0, value.get(i)); //Cada
kilómetro recorrido tiene un coste de 1 céntimo
                    goal += DatosEjercicio4.getBeneficio(value.get(i)) - fmAux;
                    //Cada kilómetro recorrido tiene un coste de 1 céntimo
                }
                else { //si la arista no existe, la solucion es mala
                    restriccion++;
                }
            } else {
                if (DatosEjercicio4.existeArista(value.get(i - 1), value.get(i))) {
                    fmAux += DatosEjercicio4.getDistancia(value.get(i - 1),
value.get(i));
                    goal += DatosEjercicio4.getBeneficio(value.get(i)) - fmAux;
                } else { //si la arista no existe, la solucion es mala
                    restriccion++;
                }
            }
        }

        if (value.get(value.size() - 1) != 0) { //si no vuelve al almacén no sirve
            restriccion = restriccion * 2;
        }
    }
}

```

```

        //Fitness Maximo
        fmAux = 0.;
        for (int i = 0; i < value.size(); i++) {
            fmAux += DatosEjercicio4.getBeneficio(value.get(i)); //Maximizar beneficio
        }
        fM = Math.pow(fmAux, 2);

        return 0.0 + goal - fM * restriccion;
    }

    @Override
    public SolucionEjercicio4 solucion(List<Integer> value) {
        // TODO Auto-generated method stub
        return SolucionEjercicio4.of_Range(value);
    }

    @Override
    public Integer itemsNumber() {
        // TODO Auto-generated method stub
        return DatosEjercicio4.getNumVertices();
    }
}

```


TestEj4AGRange.java

```
package ejercicio4;

import java.util.List;
import java.util.Locale;

import _soluciones.SolucionEjercicio4;
import us.lsi.ag.agchromosomes.AlgoritmoAG;
import us.lsi.ag.agstopping.StoppingConditionFactory;

public class TestEj4AGRange {

    public static void main(String[] args) {
        Locale.setDefault(new Locale("en", "US"));

        AlgoritmoAG.ELITISM_RATE = 0.10;
        AlgoritmoAG.CROSSOVER_RATE = 0.95;
        AlgoritmoAG.MUTATION_RATE = 0.8;
        AlgoritmoAG.POPULATION_SIZE = 1000;

        StoppingConditionFactory.NUM_GENERATIONS = 1000;
        StoppingConditionFactory.stoppingConditionType =
        StoppingConditionFactory.StoppingConditionType.GenerationCount;

        for (int i = 1; i < 3; i++) {
            Ejercicio4AG p = new Ejercicio4AG("ficheros/Ejercicio4DatosEntrada" + i +
            ".txt");

            AlgoritmoAG<List<Integer>, SolucionEjercicio4> ap = AlgoritmoAG.of(p);
            ap.ejecuta();
            System.out.println("=====");
            System.out.println(ap.bestSolution());
            System.out.println("=====\\n");
        }
    }
}
```

Resultados AG

```
[Cliente[id=0, beneficio=0.0], Cliente[id=1, beneficio=400.0], Cliente[id=2, beneficio=300.0], Cliente[id=3, beneficio=200.0], Cliente[id=4, beneficio=100.0]]
[Conexion[id=0, distancia=1.0], Conexion[id=1, distancia=100.0], Conexion[id=2, distancia=1.0], Conexion[id=3, distancia=100.0], Conexion[id=4, distancia=1.0],
Conexion[id=5, distancia=1.0], Conexion[id=6, distancia=100.0], Conexion[id=7, distancia=5.0]]
=====
Camino a seguir:
[0, 1, 2, 3, 4, 0]
Distancia: 9.0
Beneficio: 981.0
=====

[Cliente[id=0, beneficio=0.0], Cliente[id=1, beneficio=100.0], Cliente[id=2, beneficio=200.0], Cliente[id=3, beneficio=300.0], Cliente[id=4, beneficio=200.0],
Cliente[id=5, beneficio=300.0], Cliente[id=6, beneficio=200.0], Cliente[id=7, beneficio=200.0]]
[Conexion[id=8, distancia=2.0], Conexion[id=9, distancia=1.0], Conexion[id=10, distancia=1.0], Conexion[id=11, distancia=3.0], Conexion[id=12, distancia=1.0],
Conexion[id=13, distancia=1.0], Conexion[id=14, distancia=3.0], Conexion[id=15, distancia=1.0], Conexion[id=16, distancia=1.0], Conexion[id=17, distancia=3.0],
Conexion[id=18, distancia=1.0], Conexion[id=19, distancia=1.0], Conexion[id=20, distancia=1.0]]
=====
Camino a seguir:
[0, 2, 5, 3, 7, 4, 6, 1, 0]
Distancia: 9.0
Beneficio: 1463.0
=====
```