ADDA - Practica Individual 1
Problema 3

Juan Arteaga Carmona
(juan.arteaga41567@gmail.com)
2º Curso - TI-2

5 de abril de 2018

1. Complete la ficha de descripción del problema

- Tipos:
  S - List<String>

- Propiedades compartidas:
  N - Integer - Número total de jugadores
  M - Integer - Presupuesto
  S - Integer - Número de jugadores que hay que seleccionar
  LJ - List<Jugador>- Lista de jugadores disponibles

- Solucion:
  Seleccionar S jugadores de entre la lista de jugadores de forma que se optimice la suma de los tiros cortos y largos teniendo en cuenta que no podemos sobrepasar el presupuesto M, se tienen que cubrir al menos 2 puestos de pivots, 3 de aleros y que debe de haber tan solo un jugador que pueda jugar como base.

- Propiedades:
  x_i - Jugador i
  VC_i - Valor de los tiros cortos del jugador i
  VL_i - Valor de los tiros largos del jugador i
  CA_i - Cache del jugador
  $JisBASE_i$ - Indicador de posicion de base para el jugador i
  $JisPIV_i$ - Indicador de posicion de pivot para el jugador i
  $JisALE_i$ - Indicador de posicion de alero para el jugador i

- Restricciones:

  La suma de los caches de los jugadores seleccionados no puede superar el presupuesto del entrenador.

$$\sum_{i \in [0,N)} x_i \, CA_i \leq M \tag{1}$$

Debemos de seleccionar un numero S de jugadores de entre los disponibles.

$$\sum_{i \in [0,N)} x_i = S \tag{2}$$

Se debe de seleccionar un jugador que pueda jugar como base.

$$\sum_{i \in [0,N)} x_i \, JisBASE_i = 1 \tag{3}$$

Se deben de seleccionar al menos 3 aleros

$$\sum_{i \in [0,N)} x_i \, JisALE_i \geq 3 \tag{4}$$

Se deben de seleccionar al menos 2 pivots

$$\sum_{i \in [0,N)} x_i \, JisPIV_i \geq 2 \tag{5}$$

- Solución óptima:

$$max \sum_{i \in [0,N)} x_i \, VC_i + \sum_{i \in [0,N)} x_i \, VL_i$$

2. Resolver el problema por Programacion lineal o Programacion linea entera, para ello:

2.1. Indique razonadamente si es adecuado usar PL o PLI

Dado que estamos tratando con datos que son números enteros, podemos afirmar que el uso de PLI es el adecuado. De hecho, si usasemos PL sería posible que obteniesemos soluciones no válidas, como por ejemplo que sólo se seleccione la mitad de un jugador.

2.2. Completar la ficha de descripción de la solucion mediante la programación lineal. Justifique porque ha incluido cada variable y cada restricción.

- Propiedades compartidas:
  N - Integer - Numero total de jugadores
  M - Integer - Presupuesto
  S - Integer - Numero de jugadores que hay que seleccionar
  LJ - List<Jugador>- Lista de jugadores disponibles

- Variables:
  x_i - Jugador i
  VC_i - Valor de los tiros cortos del jugador i
  VL_i - Valor de los tiros largos del jugador i

CA_i - Cache del jugador
$JisBASE_i$ - Indicador de posicion de base para el jugador i
$JisPIV_i$ - Indicador de posicion de pivot para el jugador i
$JisALE_i$ - Indicador de posicion de alero para el jugador i

- Restricciones:

$$\sum_{i \in [0,N]} x_i \, CA_i \leq M \tag{1}$$

$$\sum_{i \in [0,N]} x_i = S \tag{2}$$

$$\sum_{i/x_i.getPos1==\text{"}Base\text{"}|x_i.getPos2==\text{"}Base\text{"}} x_i = 1 \tag{3}$$

$$\sum_{i/x_i.getPos1==\text{"}Alero\text{"}|x_i.getPos2==\text{"}Alero\text{"}} x_i \geq 3 \tag{4}$$

$$\sum_{i/x_i.getPos1==\text{"}Pivot\text{"}|x_i.getPos2==\text{"}Pivot\text{"}} x_i \geq 2 \tag{5}$$

- Función objetivo:

$$max \sum_{i \in [0,N)} x_i \, VC_i + \sum_{i \in [0,N)} x_i \, VL_i$$

2.3. Genere un archivo denominado 'suplentes.txt' con los datos del escenario de entrada de forma similar a como se ha realizado en las clases de prácticas para otros problemas

Este archivo es un volcado directo de los datos presentados en el enunciado en formato CSV. Una vez se ejecute el programa estas lineas serán leidas y se creará una lista de jugadores que representan nuestro datos iniciales.

```
0,Alex,Alero,Escolta,1,España,2,5,1
1,Carlos,Ala-Pivot,Pivot,4,España,4,4,4
2,Jordi,Pivot,Ala-Pivot,3,España,5,3,3
3,Victor,Escolta,Ala-Pivot,1,España,1,3,1
4,Fran,Ala-Pivot,Escolta,2,España,2,5,2
5,Michael,Base,Escolta,3,USA,3,3,5
6,Drazen,Pivot,Escolta,1,Croacia,2,1,4
7,Emanuel,Base,Pivot,2,Argentina,2,3,2
8,Toni,Alero,Pivot,2,Croacia,2,5,2
9,Yao,Ala-Pivot,Alero,3,Francia,3,3,3
10,Pablo,Base,Escolta,4,Argentina,4,4,4
11,Dino,Pivot,Pivot,2,Croacia,2,2,2
12,Lamarcus,Base,Ala-Pivot,2,USA,2,2,2
13,Mark,Alero,Pivot,1,USA,1,5,3
14,Juan,Base,Base,3,Argentina,3,3,3
```

```
15,Homero,Pivot,Ala-Pivot,4,Argentina,4,2,4
16,Chris,Base,Base,5,USA,5,5,5
17,Joseph,Ala-Pivot,Escolta,1,Francia,1,5,3
18,Zoran,Pivot,Alero,2,Croacia,4,3,2
19,Laurent,Base,Escolta,3,Francia,3,3,3
```

2.4. Desarolle un proyecto que resuelva el problema especificado por la técnica indicada. 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.

El código del proyecto se puede consultar en los anexos.

2.5. Dicho proyecto debe incluir un test de prueba que genere la solución para el escenario previamente descrito. Debe entregar tanto el archivo en formato LPSolve generado, como la solución obtenida para dicho escenario.

El test de prueba de programación lineal se encuentra en dentro de la clase ProblemaBaloncestoPLI. En esta clase se lee el archivo suplentes.txt, se crea un archivo LPSolve y finalmente se ejecuta.

- Archivo con formato LPsolve:

```
max: 6*x0 + 8*x1 + 6*x2 + 4*x3 + 7*x4 + 8*x5 + 5*x6 + 5*x7 + 7*x8 + 6*x9 + 8*x10
 ↪  + 4*x11 + 4*x12 + 8*x13 + 6*x14 + 6*x15 + 10*x16 + 8*x17 + 5*x18 + 6*x19;

1*x0+4*x1+3*x2+1*x3+2*x4+3*x5+1*x6+2*x7+2*x8+3*x9+4*x10+2*x11+2*x12+1*x13+3*x14+4*x15+5*x16+1*x
 ↪  <= 10;
x0+x1+x2+x3+x4+x5+x6+x7+x8+x9+x10+x11+x12+x13+x14+x15+x16+x17+x18+x19 = 7;
x5+x7+x10+x12+x14+x16+x19+0 = 1;
x1+x2+x6+x7+x8+x11+x13+x15+x18+0 >= 2;
x0+x8+x9+x13+x18+0 >= 3;

bin x0 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16 x17 x18 x19 ;
```

- Solucion obtenida:

```
[Alex, Fran, Drazen, Emanuel, Toni, Mark, Joseph]
```

Como curiosidad, alterando el orden de las restriciones tambien obtenemos otros resultados, que son válidos, pero no son el mismo que se especifica en el enunciado. En los anexos se puede ver un volcado de pantalla de la solucion del enunciado

3. Resolver el problema mediante algoritmo genético, para ello:

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

En este problema debemos de seleccionar un numero de jugadores entre unos disponibles. Por lo tanto, podriamos utilizar cualquier tipo de cromosoma de tipo valores (ValueInRangeProblem). Aun así, tan solo se utilizaran los valores 0 y 1, por lo que el cromosoma ideal sería el de valores binarios. De hecho, si no se utilizase este cromosoma se debería de controlar que los genes fuesen solo 0 o 1 para que no se diesen soluciones no validas como por ejemplo que solo se seleccione la mitad de un jugador o mas de una vez un mismo jugador.

3.2. Complete la ficha de desecripción de la solucion mediante algoritmo genético.

- S:
- E: Integer
- Propiedades Compartidas:
  N - Integer - Número total de jugadores
  M - Integer - Presupuesto
  S - Integer - Número de jugadores que hay que seleccionar
  LJ - List<Jugador>- Lista de jugadores disponibles
- Tipo de cromosoma: Binario
- Decode:
  - d: List<Integer>
  - d.size: S
  - $d_i \in [0, 1]$ - Ya que se trata de un cromosoma binario
- Fitness: $V - r^2\, k$
  - V: $\sum_{i/d_i==1} (VC_i + VL_i)$
  - r = $\sum_0^4 r_i$
  - $r_0 = \begin{cases} k & \text{si } \sum d_i \neq S \\ 0 & \text{e.c.o.c} \end{cases}$
  - $r_1 = \begin{cases} \sum d_i * d_i.getCache() & \text{si } \sum d_i * d_i.getCache() > M \\ 0 & \text{e.c.o.c} \end{cases}$
  - $r_2 = \begin{cases} \sum_{i/d_i.getPos1()\mid d_i.getPos2()==\text{"}Base\text{"}} d_i & \text{si } \sum_{i/d_i.getPos1()\mid d_i.getPos2()==\text{"}Base\text{"}} d_i \neq 1 \\ 0 & \text{e.c.o.c} \end{cases}$

- $r_3 = \begin{cases} \sum_{i/d_i.getPos1()|d_i.getPos2()==\text{"}Alero\text{"}} d_i & \text{si } \sum_{i/d_i.getPos1()|d_i.getPos2()==\text{"}Alero\text{"}} d_i < 2 \\ 0 & \text{e.c.o.c} \end{cases}$
- $r_4 = \begin{cases} \sum_{i/d_i.getPos1()|d_i.getPos2()==\text{"}Pivot\text{"}} d_i & \text{si } \sum_{i/d_i.getPos1()|d_i.getPos2()==\text{"}Pivot\text{"}} d_i < 3 \\ 0 & \text{e.c.o.c} \end{cases}$
- k: $10 * TAMjugadores$ - Teniendo en cuenta que el maximo del valor de los tiros puede es 5 (10 al ser largos y cortos).

▪ : Solucion:

- List<String>s = new ArrayList<String>();
- range(0,S).foreach(i ->s.add(jugadores.get(i).getNombre()));

3.3. Desarrolle un proyecto que resuelva el problema especificado por la tecnica indicada. Tenga en cuenta que debe dar una implementación general que genere la ssilucion requerida para cualquier problema de entrada, y no solo para el escenario concreto que se proporciona en este enunciado.

Al igual que con la parte de PLI, es posible consultar el código completo en el anexo. La solución al problema con algoritmo genético seran las clases ProblemaBaloncestoAG.java y TestAG.java, en la primera se construye el tipo y en la segunda se testea sobre el ejemplo propuesto en el enunciado.

3.4. Complete el test de prueba e indique qué solucion obtiene para el problema propuesto en el enunciado. Los datos del problema se facilitan en el fichero "suplentes.txt".

Debido a que el algoritmo genético es un algoritmo de aproximación, es posible que al ejecutarlo se den resultados que no sean óptimos o, en nuestro caso, exactamente el mismo que vemos en el ejemplo. Sin embargo, en las pruebas realizadas se ha conseguido asegurar la solución del enunciado al utilizar un tamaño de poblacion de 10000 y 500 generaciones en el 75 % de las ocasiones.
En el anexo se puede ver el volcado de pantalla de la solución obtenida al ejecutar la clase TestAG

4. Anexos

4.1. Codigo completo

4.1.1. Clase MetodosAuxiliares

```
package andalu30.PracticaIndividual1;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.util.ArrayList;
```

```
import java.util.List;

public class MetodosAuxiliares {

	public static List<Jugador> getJugadoresDesdeArchivo(String path){
		List<Jugador> res = new ArrayList<>();

		try {
			File archivo = new File(path);
			FileReader fr = new FileReader(archivo);
			BufferedReader br = new BufferedReader(fr);

			// Lectura del fichero
		String linea;
		String[] div = null;
			while((linea=br.readLine())!=null) {
			div = linea.split(",");
				res.add(new Jugador(new Integer(div[0]),
				 ↪  div[1],div[2],div[3],new
				 ↪  Integer(div[4]),div[5],new Integer(div[6]),new
				 ↪  Integer(div[7]),new Integer(div[8])));
			}
			fr.close();

		} catch (Exception e) {
			System.err.println("Se ha procucido un error al abrir el
			 ↪  archivo especificado en el path. "+e.getMessage());
		}

		return res;
	}
}
```

4.1.2. Clase Jugador

```
package andalu30.PracticaIndividual1;

public class Jugador {
	int id;
	String nombre;
	String pos1;
	String pos2;
	int cache;
	String nacion;
	int minJugados;
	int valorCortos;
	int valorLargos;



	public Jugador(int id, String nombre, String pos1, String pos2, int cache,
	 ↪  String nacion, int minJugados,
			int valorCortos, int valorLargos) {
		super();
		this.id = id;
		this.nombre = nombre;
		this.pos1 = pos1;
		this.pos2 = pos2;
```

```
        this.cache = cache;
        this.nacion = nacion;
        this.minJugados = minJugados;
        this.valorCortos = valorCortos;
        this.valorLargos = valorLargos;
}


public int getId() {
        return id;
}


public String getNombre() {
        return nombre;
}


public String getPos1() {
        return pos1;
}


public String getPos2() {
        return pos2;
}


public int getCache() {
        return cache;
}


public String getNacion() {
        return nacion;
}


public int getMinJugados() {
        return minJugados;
}


public int getValorCortos() {
        return valorCortos;
}


public int getValorLargos() {
```

```
        return valorLargos;
}


@Override
public int hashCode() {
        final int prime = 31;
        int result = 1;
        result = prime * result + cache;
        result = prime * result + id;
        result = prime * result + minJugados;
        result = prime * result + ((nacion == null) ? 0 : nacion.hashCode());
        result = prime * result + ((nombre == null) ? 0 : nombre.hashCode());
        result = prime * result + ((pos1 == null) ? 0 : pos1.hashCode());
        result = prime * result + ((pos2 == null) ? 0 : pos2.hashCode());
        result = prime * result + valorCortos;
        result = prime * result + valorLargos;
        return result;
}


@Override
public boolean equals(Object obj) {
        if (this == obj)
                return true;
        if (obj == null)
                return false;
        if (getClass() != obj.getClass())
                return false;
        Jugador other = (Jugador) obj;
        if (cache != other.cache)
                return false;
        if (id != other.id)
                return false;
        if (minJugados != other.minJugados)
                return false;
        if (nacion == null) {
                if (other.nacion != null)
                        return false;
        } else if (!nacion.equals(other.nacion))
                return false;
        if (nombre == null) {
                if (other.nombre != null)
                        return false;
        } else if (!nombre.equals(other.nombre))
                return false;
        if (pos1 == null) {
                if (other.pos1 != null)
                        return false;
        } else if (!pos1.equals(other.pos1))
                return false;
        if (pos2 == null) {
                if (other.pos2 != null)
                        return false;
        } else if (!pos2.equals(other.pos2))
                return false;
```

```
                if (valorCortos != other.valorCortos)
                        return false;
                if (valorLargos != other.valorLargos)
                        return false;
                return true;
        }



        @Override
        public String toString() {
                return "Jugador [id=" + id + ", nombre=" + nombre + ", pos1=" + pos1 +
                 ↪   ", pos2=" + pos2 + ", cache=" + cache
                                + ", nacion=" + nacion + ", minJugados=" + minJugados
                                 ↪   + ", valorCortos=" + valorCortos
                                + ", valorLargos=" + valorLargos + "]\n";
        }




}
```

4.1.3. Clase ProblemaBaloncestoPLI

```
package andalu30.PracticaIndividual1;

import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.List;
import us.lsi.algoritmos.Algoritmos;
import us.lsi.pl.AlgoritmoPLI;

public class ProblemaBaloncestoPLI {

        public static void main(String[] args) {

                Integer presupuesto = 10;
                Integer seleccionarJugadores = 7;

                List<Jugador> jugadores =
                 ↪   MetodosAuxiliares.getJugadoresDesdeArchivo("ficheros/suplentes.txt");

                //Inicializacion de la string
                String r = "";
                r = r+"max: ";

                //Funcion objetivo. Variables y pesos
                for(int i =0;i<jugadores.size();i++){
                        if (i!=0) r +=" + ";
                        r +=
                         ↪   jugadores.get(i).getValorCortos()+jugadores.get(i).getValorLargos()
                         ↪   + "*x"+i;
                }
                r += ";\n\n";
```

```
//Restricciones.
//Suma de cache<= presupuesto
                for (int i = 0; i < jugadores.size(); i++) {
                        if (i!=0) r = r+"+";
                        r += jugadores.get(i).getCache()+"*x" + i;
                }
                r+= " <= "+presupuesto+";\n";

//Seleccionar solamente n jugadores
for (int i = 0; i < jugadores.size(); i++) {
        if (i!=0) r = r+"+";
        r += "x" + i;
}
r += " = "+seleccionarJugadores+";\n";

//Solo un base
for (int i = 0; i < jugadores.size(); i++) {
        if (jugadores.get(i).getPos1().equals("Base")||
        ↪  jugadores.get(i).getPos2().equals("Base")) {
                r += "x"+i+"+";
        }
}
r += "0 = 1;\n";


//Pivots
for (int i = 0; i < jugadores.size(); i++) {
        if( jugadores.get(i).getPos1().equals("Pivot") ||
        ↪  jugadores.get(i).getPos2().equals("Pivot")) {
                r += "x"+i+"+";
        }
}
r += "0 >= 2;\n";


//Aleros
for (int i = 0; i < jugadores.size(); i++) {
        if( jugadores.get(i).getPos1().equals("Alero") ||
        ↪  jugadores.get(i).getPos2().equals("Alero")) {
                r += "x"+i+"+";
        }
}
r += "0 >= 3;\n\nbin ";

//Declaracion variables binarias
for (int i = 0; i < jugadores.size(); i++) {
        r+="x"+i+" ";
}
r+=";";
//------------------------------------------------
//Guardar la string a un archivo

try {
        PrintWriter out = new
        ↪  PrintWriter("ficheros/ArchivoLPSolveGenerado.txt");
        out.print(r);
        out.close();
} catch (Exception e) {
```

```
                System.err.println("Se ha producido un error al guardar el
                 ↪  archivo LPSolve generado. "+e.getMessage()+"\n A
                 ↪  continuacion se imprime el texto generado que deberia de
                 ↪  haber sido guardado en ese archivo.\n"+r);
            }

//-------------------------------------------------------------------


            AlgoritmoPLI a =
             ↪  Algoritmos.createPLI("ficheros/ArchivoLPSolveGenerado.txt");

            List<String> NombresSolucion = new ArrayList<>();

            a.ejecuta();

            double[] solucion = a.getSolucion();
            for (int i=0;i<solucion.length;i++) {
                    if (solucion[i]==1.) {
                            NombresSolucion.add(jugadores.get(i).getNombre());
                    }
            }


            //Impresiones por pantalla:
            System.out.println("Se ha generado un archivo llamado
             ↪  \"ArchivoLPSolveGenerado.txt\" que contiene la definicion de la
             ↪  solucion del problema en formato LPSolve");
            System.out.println("Este es el archivo con formato LPSolve:\n\n"+r);
            System.out.println("\n\nUna vez ejecutado, esta es la solucion al
             ↪  problema (tambien se guardara en un archivo de texto):");
            System.out.println(NombresSolucion);


            try {
                    PrintWriter out = new PrintWriter("ficheros/Solucion.txt");
                    out.print(NombresSolucion);
                    out.close();
            } catch (Exception e) {
                    System.err.println("Se ha producido un error al realizar la
                     ↪  escritura de la solucion en un archivo. "+e.getMessage()+"
                     ↪  Esta ha sido la solucion:\n"+NombresSolucion);
            }


        }

}
```

4.1.4. Clase ProblemaBaloncestoAG

```
package andalu30.PracticaIndividual1;

import java.util.List;
import us.lsi.ag.ValuesInRangeChromosome;
import us.lsi.ag.ValuesInRangeProblemAG;
```

```java
public class ProblemaBaloncestoAG implements ValuesInRangeProblemAG<Integer,
 ↪  List<Integer>> {
        private List<Jugador> jugadores;
        private Integer  presupuesto = 10;
        private Integer seleccionarJugadores = 7;

        public static ProblemaBaloncestoAG create(String fichero) {
                return new ProblemaBaloncestoAG(fichero);
        }

        private ProblemaBaloncestoAG(String fichero) {
                this.jugadores = MetodosAuxiliares.getJugadoresDesdeArchivo(fichero);
        }


        @Override
        public Integer getVariableNumber() {
                return jugadores.size();
        }


        //Al ser binario no deberia de hacer falta, pero me obliga a usarlo...
        @Override
        public Integer getMax(Integer i) {
                return 1;
        }

        @Override
        public Integer getMin(Integer i) {
                return 0;
        }

        @Override
        public Double fitnessFunction(ValuesInRangeChromosome<Integer> cr) {
                Double v = 0.;
                Double k = 200.;

                //Numero de jugadores
                int contAux = 0;
                for (Integer i : cr.decode()){
                        if (i==1) {
                                contAux++;
                        }
                }
                if (contAux!=seleccionarJugadores) {
                        v=-1000000.;
                }

                //V
                for (int i = 0; i < cr.decode().size(); i++) {
                        if (cr.decode().get(i)==1) {
                                v +=
                                 ↪  jugadores.get(i).getValorCortos()+jugadores.get(i).getValorLargo
                        }
                }

                //Presupuesto
                Double r1 = 0.;
```

```
for (int i = 0; i < cr.decode().size(); i++) {
        if (cr.decode().get(i)==1) {
                r1 += jugadores.get(i).getCache();
        }
}
if (!(r1 > presupuesto)) {
        r1 = 0.;
}

//Solo un base
Double r2 = 0.;
for (int i = 0; i < cr.decode().size(); i++) {
        if (cr.decode().get(i)==1 &&
         ↪  (jugadores.get(i).getPos1().equals("Base") ||
         ↪  jugadores.get(i).getPos2().equals("Base") )) {
                r2++;
        }
}
if (r2 == 1) {
        r2 = 0.;
}else {
        ;
}

//Al menos dos pivots
Double r3 = 0.;
for (int i = 0; i < cr.decode().size(); i++) {
        if (cr.decode().get(i)==1 &&
         ↪  (jugadores.get(i).getPos1().equals("Pivot") ||
         ↪  jugadores.get(i).getPos2().equals("Pivot") )) {
                r3++;
        }
}
if (r3 >= 2) {
        r3 = 0.;
}else {
        r3 = -r3;
}

//Al menos tres aleros
Double r4 = 0.;
for (int i = 0; i < cr.decode().size(); i++) {
        if (cr.decode().get(i)==1 &&
         ↪  (jugadores.get(i).getPos1().equals("Alero") ||
         ↪  jugadores.get(i).getPos2().equals("Alero") )) {
                r4++;
        }
}
if (r4 >= 0) {
        r4 = 0.;
}else {
        r4 = -r4;
}

Double r = r1+r2+r3+r4;

Double fitness = v - (r*k);
//System.out.println(fitness);
```

```
                return fitness;
        }

        @Override
        public List<Integer> getSolucion(ValuesInRangeChromosome<Integer> cr) {
                return cr.decode();
        }


}
```

4.1.5. Clase TestAG

```
package andalu30.PracticaIndividual1;

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

import us.lsi.ag.ValuesInRangeChromosome;
import us.lsi.ag.agchromosomes.AlgoritmoAG;
import us.lsi.ag.agchromosomes.ChromosomeFactory;
import us.lsi.ag.agchromosomes.ChromosomeFactory.ChromosomeType;
import us.lsi.ag.agstopping.StoppingConditionFactory;


public class TestAG {


        public static void main(String[] args){

                System.out.println("Ejecutando el algoritmo genetico. Por favor,
                 ↪  espere.");

                AlgoritmoAG.ELITISM_RATE  = 0.3;
                AlgoritmoAG.CROSSOVER_RATE = 0.8;
                AlgoritmoAG.MUTATION_RATE = 0.7;
                AlgoritmoAG.POPULATION_SIZE = 1000;

                StoppingConditionFactory.NUM_GENERATIONS = 500;
                StoppingConditionFactory.FITNESS_MIN = 50.;


                ProblemaBaloncestoAG p =
                 ↪  ProblemaBaloncestoAG.create("ficheros/suplentes.txt");

                AlgoritmoAG  ap = new AlgoritmoAG(ChromosomeType.Binary, p);

                ap.ejecuta();

                System.out.println("Algoritmo genetico ejecutado. Estas son las
                 ↪  estadisticas:");

                ValuesInRangeChromosome<Integer> cr =
                 ↪  ChromosomeFactory.asValuesInRange(ap.getBestFinal());
                System.out.println("Fitness del mejor cromosoma:      "+
                 ↪  p.fitnessFunction(cr));
```

```
                System.out.println("Solucion del algoritmo genetico:
                ↪  "+p.getSolucion(cr));

                List<Jugador> jugadores =
                ↪  MetodosAuxiliares.getJugadoresDesdeArchivo("ficheros/suplentes.txt");
                List<String> nombresSolucion = new ArrayList<>();

                for (int i=0; i<cr.decode().size();i++) {
                        if (cr.decode().get(i).equals(1)) {
                                nombresSolucion.add(jugadores.get(i).getNombre());
                        }
                }
                System.out.println("\nInterpretacion de la solucion:\nSe deben de
                ↪  seleccionar los siguientes jugadores: "+nombresSolucion);

        }

}
```

4.2. Volcado de pantalla de los resultados obtenidos por cada prueba realizada

```
Se ha generado un archivo llamado "ArchivoLPSolveGenerado.txt" que contiene
la definición de la solución del problema en formato LPSolve
Este es el archivo con formato LPSolve:

max: 6*x0 + 8*x1 + 6*x2 + 4*x3 + 7*x4 + 8*x5 + 5*x6 + 5*x7 + 7*x8 + 6*x9 +
8*x10 + 4*x11 + 4*x12 + 8*x13 + 6*x14 + 6*x15 + 10*x16 + 8*x17 + 5*x18 +
6*x19;

1*x0+4*x1+3*x2+1*x3+2*x4+3*x5+1*x6+2*x7+2*x8+3*x9+4*x10+2*x11+2*x12+1*x13+3
*x14+4*x15+5*x16+1*x17+2*x18+3*x19 <= 10;
x0+x1+x2+x3+x4+x5+x6+x7+x8+x9+x10+x11+x12+x13+x14+x15+x16+x17+x18+x19 = 7;
x5+x7+x10+x12+x14+x16+x19+0 = 1;
x1+x2+x6+x7+x8+x11+x13+x15+x18+0 >= 2;
x0+x8+x9+x13+x18+0 >= 3;

bin x0 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16 x17 x18 x19 ;


Una vez ejecutado, esta es la solución al problema (tambien se guardará en
un archivo de texto):
[Alex, Fran, Drazen, Emanuel, Toni, Mark, Joseph]
```

Figura 1: Volcado de pantalla de la terminal al ejecutar la clase ProblemaBaloncestoPLI

```
Ejecutando el algoritmo genetico. Por favor, espere.
Algoritmo genetico ejecutado. Estas son las estadisticas:
Fitness del mejor cromosoma:      46.0
Solucion del algoritmo genetico: [1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0]

Interpretacion de la solucion:
Se deben de seleccionar los siguientes jugadores: [Alex, Fran, Drazen, Emanuel, Toni, Mark, Joseph]
```

Figura 2: Volcado de pantalla de la terminal al ejecutar la clase TestAG

4.3. Código fuente y licencia

Todo el código fuente del trabajo se podrá encontrar en `www.github.com/Andalu30/US-ADDA-PracticaIndividual1/` a partir del lunes 9 de abril de 2018, fecha limite de entrega de este trabajo. No se recomienda el uso de la totalidad o de parte de este trabajo en caso de que no se cambie la actividad en años posteriores debido a que la universidad usa software de detección de copias. El contenido de este trabajo es libre y se encuentra licenciado bajo una licencia GPL v3.

```
This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program.  If not, see <http://www.gnu.org/licenses/>.
```