# PROBLEMA DE LOS SUPLENTES

*David Gil Baeza*

# 1 - Ficha de descripción del problema

**ANÁLISIS Y DISEÑO DE DATOS Y ALGORITMOS. CURSO 2017/18.**
**FICHA PARA DEFINICIÓN DE PROBLEMAS**

***Descripción del Problema*** Suplentes

***Tipos***

- S – SolucionSupl

***Propiedades Compartidas***

L: List<Jugador>
N: nº de jugadores
M: Presupuesto
S: nº de suplentes a contratar.
$tc_i$: tiros cortos de i.
$tl_i$: tiros largos de i
$c_i$: cachés de i
$P1_i$: Posición 1 de i
$P2_i$: Posición 2 de i

***Solución:*** Los suplentes $x_i$ que debemos elegir de L para así poder maximizar el número de suplentes a contratar sin superar el presupuesto, habiendo como mínimo 2 pivotes y 3 aleros, además de un base. Hay que maximizar la suma de tiros cortos y largos.

***Propiedades:***

$x_i$: Variable binaria para indicar si se contrata al jugador i.

***Restricciones:***

$R_0: \sum_{i=0}^{m-1} x_i = S$

$R_1: \sum_{i=0}^{m-1} x_i \geq 2$ (jugadores "pivote")

$R_2: \sum_{i=0}^{m-1} x_i \geq 3$ (jugadores "alero")

$R_3: \sum_{i=0}^{m-1} x_i \cdot c_i \leq M$

$R_4: \sum_{i=0}^{m-1} x_i = 1$ (jugadores "base")

***Solución óptima:***

$$\text{máx} \sum_{i=0}^{m-1} (x_i \cdot tc_i + x_i \cdot tl_i)$$

# * - Implementación: Código

## - *Interfaz "Jugador"*

```
package problemaSuplentes.tipos;

public interface Jugador {
String getNombre();
void setNombre(String nombre);
Posicion getPos1();
void setPos1(Posicion pos1);
Posicion getPos2();
void setPos2(Posicion pos2);
Integer getCache();
void setCache(Integer cache);
String getNacion();
void setNacion(String nacion);
Integer getMinJugados();
void setMinJugados(Integer minJugados);
Integer getTirosCortos();
void setTirosCortos(Integer tirosCortos);
Integer getTirosLargos();
void setTirosLargos(Integer tirosLargos);

}
```

## - Clase "JugadorImpl"

```
package problemaSuplentes.tipos;

public class JugadorImpl implements Jugador {

	private String nombre;
	private Posicion pos1;
	private Posicion pos2;
	private Integer cache;
	private String nacion;
	private Integer minJugados;
	private Integer tirosCortos;
	private Integer tirosLargos;

	public int hashCode() {
		final int prime = 31;
		int result = 1;
		result = prime * result + ((cache == null) ? 0 :
cache.hashCode());
		result = prime * result + ((minJugados == null) ? 0 :
minJugados.hashCode());
		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 + ((tirosCortos == null) ? 0 :
tirosCortos.hashCode());
            result = prime * result + ((tirosLargos == null) ? 0 :
tirosLargos.hashCode());
            return result;
      }

      public boolean equals(Object obj) {
            if (this == obj)
                  return true;
            if (obj == null)
                  return false;
            if (getClass() != obj.getClass())
                  return false;
            JugadorImpl other = (JugadorImpl) obj;
            if (cache == null) {
                  if (other.cache != null)
                        return false;
            } else if (!cache.equals(other.cache))
                  return false;
            if (minJugados == null) {
                  if (other.minJugados != null)
                        return false;
            } else if (!minJugados.equals(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 != other.pos1)
                  return false;
            if (pos2 != other.pos2)
                  return false;
            if (tirosCortos == null) {
                  if (other.tirosCortos != null)
                        return false;
            } else if (!tirosCortos.equals(other.tirosCortos))
                  return false;
            if (tirosLargos == null) {
                  if (other.tirosLargos != null)
                        return false;
            } else if (!tirosLargos.equals(other.tirosLargos))
                  return false;
            return true;
      }

      public String toString() {
            return "(" + nombre + "), Posiciones: " + pos1 + " y " +
pos2;
      }

      public JugadorImpl(Posicion pos1, Posicion pos2,
                  Integer cache, Integer tirosCortos,
                  Integer tirosLargos) {
            this.nombre = null;
```

```
        this.pos1 = pos1;
        this.pos2 = pos2;
        this.cache = cache;
        this.nacion = null;
        this.minJugados = null;
        this.tirosCortos = tirosCortos;
        this.tirosLargos = tirosLargos;

    }

    public JugadorImpl(String nombre, Posicion pos1, Posicion pos2,
                Integer cache, String nacion, Integer minJugados,
                Integer tirosCortos, Integer tirosLargos) {
        this.nombre = nombre;
        this.pos1 = pos1;
        this.pos2 = pos2;
        this.cache = cache;
        this.nacion = nacion;
        this.minJugados = minJugados;
        this.tirosCortos = tirosCortos;
        this.tirosLargos = tirosLargos;

    }

    public JugadorImpl(String s) {
        String[] trozos = s.split("#");

        this.nombre = trozos[0].trim();
        this.pos1 = Posicion.valueOf(trozos[1].trim());
        this.pos2 = Posicion.valueOf(trozos[2].trim());
        this.cache = new Integer(trozos[3].trim());
        this.nacion = trozos[4].trim();
        this.minJugados = new Integer(trozos[5].trim());
        this.tirosCortos = new Integer(trozos[6].trim());
        this.tirosLargos = new Integer(trozos[7].trim());
    }

    public String getNombre() {
        return this.nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public Posicion getPos1() {
        return this.pos1;
    }

    public void setPos1(Posicion pos1) {
        this.pos1 = pos1;
    }

    public Posicion getPos2() {
        return this.pos2;
    }

    public void setPos2(Posicion pos2) {
        this.pos2 = pos2;
    }
```

```
	public Integer getCache() {
		return this.cache;
	}

	public void setCache(Integer cache) {
		this.cache = cache;
	}

	public String getNacion() {
		return this.nacion;
	}

	public void setNacion(String nacion) {
		this.nacion = nacion;
	}

	public Integer getMinJugados() {
		return this.minJugados;
	}

	public void setMinJugados(Integer minJugados) {
		this.minJugados = minJugados;
	}

	public Integer getTirosCortos() {
		return this.tirosCortos;
	}

	public void setTirosCortos(Integer tirosCortos) {
		this.tirosCortos = tirosCortos;
	}

	public Integer getTirosLargos() {
		return this.tirosLargos;
	}

	public void setTirosLargos(Integer tirosLargos) {
		this.tirosLargos = tirosLargos;
	}

}
```

- Enumerado “Posición”

```
package problemaSuplentes.tipos;

public enum Posicion {
	Pivot, Base, Alero, AleroPivot, Escolta;
}
```

- Interfaz “SolucionSupl”

```
ppackage problemaSuplentes.tipos;

import java.util.List;
import java.util.Map;
```

```
public interface SolucionSupl {
	Map<Integer, Jugador> getMapaJugadoresSolucion();
	List<Jugador> getListaJugadoresOriginal();
	List<Jugador> getListaJugadoresSolucion();
	List<String> getNombreJugadoresSolucion();
	List<Integer> getIDJugadoresSolucion();
	String solucion();
	Integer getS();
	Integer getM();
	double getValorFuncion();

}
```

- Clase “SolucionSuplImpl”

```
package problemaSuplentes.tipos;

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

public class SolucionSuplImpl implements SolucionSupl {
	private Map<Integer, Jugador> mapaJugadores;
	private List<Jugador> listaJugadoresOriginal;
	private Integer S;
	private Integer M;
	private double valorFuncion;

	public SolucionSuplImpl (Map<Integer, Jugador>
jugadoresSolucion, List<Jugador> jugadoresOriginales,
			Integer S, Integer M, double valorFuncion) {
		this.mapaJugadores = jugadoresSolucion;
		this.listaJugadoresOriginal = jugadoresOriginales;
		this.S = S;
		this.M = M;
		this.valorFuncion = valorFuncion;
	}

	public Map<Integer, Jugador> getMapaJugadoresSolucion() {
		return this.mapaJugadores;
	}

	public List<Jugador> getListaJugadoresOriginal() {
		return this.listaJugadoresOriginal;
	}

	public List<Jugador> getListaJugadoresSolucion() {
		return new ArrayList<Jugador>
(this.getMapaJugadoresSolucion().values());
	}

	public List<String> getNombreJugadoresSolucion() {
		return
this.getMapaJugadoresSolucion().values().stream().map(x-
>x.getNombre()).collect(Collectors.toList());
	}
```

```
	@Override
	public int hashCode() {
		final int prime = 31;
		int result = 1;
		result = prime * result + ((mapaJugadores == null) ? 0 :
mapaJugadores.hashCode());
		return result;
	}

	@Override
	public boolean equals(Object obj) {
		if (this == obj)
			return true;
		if (obj == null)
			return false;
		if (getClass() != obj.getClass())
			return false;
		SolucionSuplImpl other = (SolucionSuplImpl) obj;
		if (mapaJugadores == null) {
			if (other.mapaJugadores != null)
				return false;
		} else if (!mapaJugadores.equals(other.mapaJugadores))
			return false;
		return true;
	}

	public List<Integer> getIDJugadoresSolucion() {
		return new ArrayList<Integer>
(this.getMapaJugadoresSolucion().keySet());
	}

	public String solucion() {
		String res = "Lista de jugadores completa: \n----\n";
		for(int i = 0; i < this.getListaJugadoresOriginal().size();
i++) {
			res = res + this.getListaJugadoresOriginal().get(i) +
"\n";
		}
		res = res + "----\nSuplentes que se necesitan: " +
this.getS().toString() + ", Presupuesto máximo: " +
this.getM().toString() + "\n----\nSOLUCIÓN: \n";
		res = res + this.toString() + "\nValor de la función
objetivo o fitness, según el caso: " + this.getValorFuncion() + "\n";
		return res;
	}

	public String toString() {
		String res = "";
		for(int i = 0; i < this.getListaJugadoresOriginal().size();
i++) {
			if(this.getMapaJugadoresSolucion().containsKey(i)) {
				res = res + i + " => " + "[" +
this.getMapaJugadoresSolucion().get(i).toString() + "]\n";
			}
		}
		return res;
	}

	public Integer getS() {
		return this.S;
	}
```

```
	public Integer getM() {
		return this.M;
	}

	public double getValorFuncion() {
		return this.valorFuncion;
	}

}
```

- Clase “FactoriaJugador”

```
package problemaSuplentes.tipos;

import java.util.Arrays;
import java.util.List;
import com.google.common.collect.Lists;
import us.lsi.stream.Stream2;

public class FactoriaJugador {

	public static Jugador create(Posicion pos1, Posicion pos2,
			Integer cache, Integer tirosCortos,
			Integer tirosLargos) {
		return new JugadorImpl(pos1, pos2, cache, tirosCortos,
tirosLargos);
	}

	public static Jugador create(String nombre, Posicion pos1,
Posicion pos2,
			Integer cache, String nacion, Integer minJugados,
			Integer tirosCortos, Integer tirosLargos) {
		return new JugadorImpl(nombre, pos1, pos2, cache, nacion,
minJugados, tirosCortos, tirosLargos);
	}

	public static List<Jugador> creaJugadores(String path) {
		List<String> lineas = Stream2.fromFile(path).toList();
		List<Jugador> jugadores = Lists.newArrayList();
		for(int i = 0; i < lineas.size(); i++) {
			jugadores.add(new JugadorImpl(lineas.get(i)));
		}
		return jugadores;
	}

	public static List<Jugador> creaJugadores() {
		Jugador j0 = create("Alex", Posicion.Alero,
Posicion.Escolta,
				1, "España", 2, 5, 1);
		Jugador j1 = create("Carlos", Posicion.AleroPivot,
Posicion.Pivot,
				4, "España", 4, 4, 4);
		Jugador j2 = create("Jordi", Posicion.Pivot,
Posicion.AleroPivot,
				3, "España", 5, 3, 3);
		Jugador j3 = create("Victor", Posicion.Escolta,
Posicion.AleroPivot,
```

```
                        1, "España", 1, 3, 1);
            Jugador j4 = create("Fran", Posicion.AleroPivot,
Posicion.Escolta,
                        2, "España", 2, 5, 2);
            Jugador j5 = create("Michael", Posicion.Base,
Posicion.Escolta,
                        3, "USA", 3, 3, 5);
            Jugador j6 = create("Drazen", Posicion.Pivot,
Posicion.Escolta,
                        1, "Croacia", 2,  1, 4);
            Jugador j7 = create("Emanuel", Posicion.Base,
Posicion.Pivot,
                        2, "Argentina", 2, 3, 2);
            Jugador j8 = create("Toni", Posicion.Alero, Posicion.Pivot,
                        2, "Croacia", 2, 5, 2);
            Jugador j9 = create("Yao", Posicion.AleroPivot,
Posicion.Alero,
                        3, "Francia", 3, 3, 3);
            Jugador j10 = create("Pablo", Posicion.Base,
Posicion.Escolta,
                        4, "Argentina", 4, 4, 4);
            Jugador j11 = create("Dino", Posicion.Pivot,
Posicion.Pivot,
                        2, "Croacia", 2, 2, 2);
            Jugador j12 = create("Lamarcus", Posicion.Base,
Posicion.AleroPivot,
                        2, "USA", 2, 2, 2);
            Jugador j13 = create("Mark", Posicion.Alero,
Posicion.Pivot,
                        1, "USA", 1, 5, 3);
            Jugador j14 = create("Juan", Posicion.Base, Posicion.Base,
                        3, "Argentina", 3, 3, 3);
            Jugador j15 = create("Homero", Posicion.Pivot,
Posicion.AleroPivot,
                        4, "Argentina", 4, 2, 4);
            Jugador j16 = create("Chris", Posicion.Base, Posicion.Base,
                        5, "USA", 5, 5, 5);
            Jugador j17 = create("Joseph", Posicion.AleroPivot,
Posicion.Escolta,
                        1, "Francia", 1, 5, 3);
            Jugador j18 = create("Zoran", Posicion.Pivot,
Posicion.Alero,
                        2, "Croacia", 4, 3, 2);
            Jugador j19 = create("Laurent", Posicion.Base,
Posicion.Escolta,
                        3, "Francia", 3, 3, 3);

            return Arrays.asList(j0, j1, j2, j3, j4, j5, j6, j7, j8,
j9, j10, j11, j12, j13, j14, j15, j16, j17, j18, j19);
      }
}
```

- Clase utilidad “Check”

```
package problemaSuplentes.utiles;

import java.util.List;
import problemaSuplentes.tipos.Jugador;
```

```
import problemaSuplentes.tipos.Posicion;
import problemaSuplentes.tipos.SolucionSupl;

public class Check {

	public static String compruebaSolucion(SolucionSupl s) {
		List<Jugador> jugadores = s.getListaJugadoresSolucion();
		String res = "";
		Boolean R0 = jugadores.size()==s.getS();
		Boolean R1 = jugadores.stream().filter(x-
>x.getPos1().equals(Posicion.Pivot) ||
				x.getPos2().equals(Posicion.Pivot)).count()>=2;
		Boolean R2 = jugadores.stream().filter(x-
>x.getPos1().equals(Posicion.Alero) ||
				x.getPos2().equals(Posicion.Alero)).count()>=3;
		Boolean R3 = jugadores.stream().mapToInt(x-
>x.getCache()).sum()<=s.getM();
		Boolean R4 = jugadores.stream().filter(x-
>x.getPos1().equals(Posicion.Base) ||
				x.getPos2().equals(Posicion.Base)).count()==1;
		res = res + "Cumplimiento de restricciones: \n----\n";
		if(R0) {
			res = res + "R0: Sí.\n";
		} else {
			res = res + "R0: No.\n";
		}
		if(R1) {
			res = res + "R1: Sí.\n";
		} else {
			res = res + "R1: No.\n";
		}
		if(R2) {
			res = res + "R2: Sí.\n";
		} else {
			res = res + "R2: No.\n";
		}
		if(R3) {
			res = res + "R3: Sí.\n";
		} else {
			res = res + "R3: No.\n";
		}
		if(R4) {
			res = res + "R4: Sí.\n\n";
		} else {
			res = res + "R4: No.\n\n";
		}

		return res;
	}

}
```

- Clase “ProblemaSuplentesPLI”

```
package problemaSuplentes.pli;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
```

```java
import java.util.Map;
import problemaSuplentes.tipos.Jugador;
import problemaSuplentes.tipos.Posicion;
import problemaSuplentes.tipos.SolucionSupl;
import problemaSuplentes.tipos.SolucionSuplImpl;
import us.lsi.pl.AlgoritmoPLI;

public class ProblemaSuplentesPLI {

	public static SolucionSupl resolver(List<Jugador> jugadores, int
S, int M) {
		AlgoritmoPLI a = AlgoritmoPLI.create();
		a.setConstraints(getConstraints(jugadores, S, M));
		a.ejecuta();
		Map<Integer, Jugador> mapa = new HashMap<> ();
		for (int i = 0; i < jugadores.size(); i++){
			double x = Math.round(a.getSolucion()[i]);
			if(x == 1) {
				mapa.put(i, jugadores.get(i));
			}
		}
		return new SolucionSuplImpl(mapa, jugadores, S, M,
Math.round(a.getObjetivo()));
	}

	@SuppressWarnings("unused")
	private static void muestraConstraints(List<Jugador> j,
			int S, int M) {
		System.out.println(getConstraints(j, S, M));
	}

	public static String getConstraints(List<Jugador> jugadores,
			int S, int M) {
		int N = jugadores.size();

		String res = "max: ";

		for(int i = 0; i < N; i++) {
			String tc =
jugadores.get(i).getTirosCortos().toString();
			String tl =
jugadores.get(i).getTirosLargos().toString();
			String var = AlgoritmoPLI.getVariable("x", i);
			res = res + tc + var + " + " + tl + var;
			if(i!=N-1) {
				res = res + " + ";
			} else {
				res = res + "; // Función objetivo\n\n";
			}
		}

		for(int i = 0; i < N; i++) {
			res = res + AlgoritmoPLI.getVariable("x", i);
			if(i!=N-1) {
				res = res + " + ";
			} else {
				res = res + " = " + S + "; // Restricción num
suplentes\n\n";
			}
		}
```

```
		List<String> varPivot = new ArrayList<> ();
		for(int i = 0; i < N; i++) {
			Posicion pos1 = jugadores.get(i).getPos1();
			Posicion pos2 = jugadores.get(i).getPos2();
			if(pos1.equals(Posicion.Pivot) ||
					pos2.equals(Posicion.Pivot)) {
				varPivot.add(AlgoritmoPLI.getVariable("x", i));
			}
		}
		if(varPivot.size()!=0) {
			for(int j = 0; j < varPivot.size(); j++) {
				if(j!=varPivot.size()-1) {
					res = res + varPivot.get(j) + " + ";
				} else {
					res = res + varPivot.get(j) + " >= 2; //
Restricción Pivot\n\n";
				}
			}
		}

		List<String> varAlero = new ArrayList<> ();
		for(int i = 0; i < N; i++) {
			Posicion pos1 = jugadores.get(i).getPos1();
			Posicion pos2 = jugadores.get(i).getPos2();
			if(pos1.equals(Posicion.Alero) ||
					pos2.equals(Posicion.Alero)) {
				varAlero.add(AlgoritmoPLI.getVariable("x", i));
			}
		}
		if(varAlero.size()!=0) {
			for(int j = 0; j < varAlero.size(); j++) {
				if(j!=varAlero.size()-1) {
					res = res + varAlero.get(j) + " + ";
				} else {
					res = res + varAlero.get(j) + " >= 3; //
Restricción Alero\n\n";
				}
			}
		}

		List<String> varBase = new ArrayList<> ();
		for(int i = 0; i < N; i++) {
			Posicion pos1 = jugadores.get(i).getPos1();
			Posicion pos2 = jugadores.get(i).getPos2();
			if(pos1.equals(Posicion.Base) ||
					pos2.equals(Posicion.Base)) {
				varBase.add(AlgoritmoPLI.getVariable("x", i));
			}
		}
		if(varBase.size()!=0) {
			for(int j = 0; j < varBase.size(); j++) {
				if(j!=varBase.size()-1) {
					res = res + varBase.get(j) + " + ";
				} else {
					res = res + varBase.get(j) + " = 1; //
Restricción Base\n\n";
				}
			}
		}

		for(int i = 0; i < N; i++) {
```

```
                  String cache =
jugadores.get(i).getCache().toString();
                  res = res + cache + AlgoritmoPLI.getVariable("x", i);
                  if(i!=N-1) {
                        res = res + " + ";
                  } else {
                        res = res + " <= " + M + "; // Restricción
presupuesto\n\n";
                  }
            }

            res = res + "bin ";
            for(int i = 0; i < N; i++) {
                  res = res + AlgoritmoPLI.getVariable("x", i);
                  if(i!=N-1) {
                        res = res + ", ";
                  } else {
                        res = res + "; // Declaración variables";
                  }
            }

            return res;
      }

}
```

- Clase “TestSuplentesPLI”

```
package problemaSuplentes.pli;

import java.util.List;
import problemaSuplentes.tipos.FactoriaJugador;
import problemaSuplentes.tipos.Jugador;
import problemaSuplentes.tipos.SolucionSupl;
import problemaSuplentes.utiles.Check;

public class TestSuplentesPLI {

      public static void main(String[] args) {

            //Creo la lista de jugadores a partir del fichero
suplentes.txt
            List<Jugador> jugadores =
FactoriaJugador.creaJugadores("ficheros/suplentes.txt");

            //Resuelvo el problema de PLI, cuya solución es de tipo
SolucionSupl para este problema
            SolucionSupl s = ProblemaSuplentesPLI.resolver(jugadores,
7, 10); //7 y 10 para el ejemplo del problema

            //Imprimo la solución gracias al método solucion() de la
clase SolucionSupl
            System.out.println("Problema resuelto por PL:\n"
                        + "--------\n"
                        + s.solucion());

            //Comprobamos la solución
            System.out.println(Check.compruebaSolucion(s));
      }
```

```
}
```

- Clase “ProblemaSuplentesAG”

```
package problemaSuplentes.ag;

import java.util.List;
import java.util.Map;
import com.google.common.collect.Lists;
import com.google.common.collect.Maps;
import us.lsi.ag.ValuesInRangeProblemAG;
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;
import
us.lsi.ag.agstopping.StoppingConditionFactory.StoppingConditionType;
import us.lsi.algoritmos.Algoritmos;
import us.lsi.ag.ValuesInRangeChromosome;
import problemaSuplentes.tipos.Jugador;
import problemaSuplentes.tipos.Posicion;
import problemaSuplentes.tipos.SolucionSupl;
import problemaSuplentes.tipos.SolucionSuplImpl;

public class ProblemaSuplentesAG implements
ValuesInRangeProblemAG<Integer, SolucionSupl> {

	private static List<Jugador> jugadores;
	private static Integer S;
	private static Integer M;

	public ProblemaSuplentesAG(List<Jugador> jugadores, Integer S,
Integer M) {
		ProblemaSuplentesAG.jugadores = jugadores;
		ProblemaSuplentesAG.S = S;
		ProblemaSuplentesAG.M = M;
	}

	public static SolucionSupl resolver(List<Jugador> jugadores,
Integer S, Integer M) {

		AlgoritmoAG.ELITISM_RATE  = 0.3;
		AlgoritmoAG.CROSSOVER_RATE = 0.8;
		AlgoritmoAG.MUTATION_RATE = 0.7;
		AlgoritmoAG.POPULATION_SIZE = 100;

		StoppingConditionFactory.NUM_GENERATIONS = 10000;
		StoppingConditionFactory.SOLUTIONS_NUMBER_MIN = 1;
		StoppingConditionFactory.FITNESS_MIN = 10000.;
		StoppingConditionFactory.stoppingConditionType =
StoppingConditionType.SolutionsNumber;

		ProblemaSuplentesAG p = new ProblemaSuplentesAG(jugadores,
S, M);
		AlgoritmoAG ap = Algoritmos.createAG(ChromosomeType.Binary,
p);
		ap.ejecuta();
```

```
            ValuesInRangeChromosome<Integer> cr =
ChromosomeFactory.asValuesInRange(ap.getBestFinal());

            return p.getSolucion(cr);
      }

      public SolucionSupl getSolucion(ValuesInRangeChromosome<Integer>
chromosome) {
            List<Integer> ls = chromosome.decode();
            Map<Integer, Jugador> m = Maps.newHashMap();
            for(int i = 0; i<ls.size(); i++) {
                  if(ls.get(i).equals(1)) {
                        m.put(i, ProblemaSuplentesAG.jugadores.get(i));
                  }
            }
            return new SolucionSuplImpl(m,
ProblemaSuplentesAG.jugadores, ProblemaSuplentesAG.S,
ProblemaSuplentesAG.M,
                        this.fitnessFunction(chromosome));
      }

      public List<Jugador>
getSolucionAux(ValuesInRangeChromosome<Integer> chromosome) {
            List<Integer> ls = chromosome.decode();
            List<Jugador> res = Lists.newArrayList();
            for(int i = 0; i<ls.size(); i++) {
                  if(ls.get(i).equals(1)) {
                        Jugador j =
ProblemaSuplentesAG.jugadores.get(i);
                        res.add(j);
                  }
            }
            return res;
      }

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

    public Integer getMax(Integer index){
            return ProblemaSuplentesAG.jugadores.size()-1;
      }

    public Integer getMin(Integer index){
            return 0;
      }

      public Double fitnessFunction(ValuesInRangeChromosome<Integer>
ls) {
            List<Jugador> solucion = this.getSolucionAux(ls);

            double V = solucion.stream().mapToDouble(x-
>x.getTirosCortos() + x.getTirosLargos()).sum();

            double K = solucion.stream().mapToDouble(x-
>Math.pow(x.getCache(), 2)).sum();

            double R0 = solucion.size();

            double R1 = solucion.stream().filter(x-
>x.getPos1().equals(Posicion.Pivot) ||
```

```
                        x.getPos2().equals(Posicion.Pivot)).count();

            double R2 = solucion.stream().filter(x-
>x.getPos1().equals(Posicion.Alero) ||
                        x.getPos2().equals(Posicion.Alero)).count();

            double R3 = solucion.stream().mapToDouble(x-
>x.getCache()).sum();

            double R4 = solucion.stream().filter(x-
>x.getPos1().equals(Posicion.Base) ||
                        x.getPos2().equals(Posicion.Base)).count();

            double var0 = (R0==S)?0:1;
            double var1 = (R1>=2)?0:1;
            double var2 = (R2>=3)?0:1;
            double var3 = (R3<=M)?0:1;
            double var4 = (R4==1)?0:1;

            double fitness = V - K*(var0 + var1 + var2 + var3 + var4);

            return fitness;
      }

}
```

- Clase "TestSuplentesAG"

```
package problemaSuplentes.ag;

import java.util.List;
import problemaSuplentes.tipos.FactoriaJugador;
import problemaSuplentes.tipos.Jugador;
import problemaSuplentes.tipos.SolucionSupl;
import problemaSuplentes.utiles.Check;

public class TestSuplentesAG {


      public static void main(String[] args){

            //Creo la lista de jugadores a partir del fichero
suplentes.txt
            List<Jugador> jugadores =
FactoriaJugador.creaJugadores("ficheros/suplentes.txt");

            //Resuelvo el problema de AG, cuya solución es de tipo
SolucionSupl para este problema
            SolucionSupl s = ProblemaSuplentesAG.resolver(jugadores, 7,
10); //7 y 10 para el ejemplo del problema

            //Imprimo la solución gracias al método solucion() de la
clase SolucionSupl
            System.out.println("Problema resuelto por AG:\n"
                  + "--------\n"
                  + s.solucion());

            //Comprobamos la solución
            System.out.println(Check.compruebaSolucion(s));
```

```
        }
}
```

## 2 - Resolución por “Programación Lineal”

Para resolver el problema he usado PLI, es decir, “Programación Lineal Entera”, ya que he visto conveniente usar variables binarias para modelar el problema. La variable *Xi* será 1 ó 0 en función de si el jugador *i* ha sido escogido o no como suplente.

**ANÁLISIS Y DISEÑO DE DATOS Y ALGORITMOS. CURSO 2017/18.**
**FICHA PARA PROBLEMAS DE PLI**

Problema PLI: Suplentes

Variables:

$x_i : [0, 1]$, binaria.

Restricciones:

$$R_0: \sum_{i=0}^{m-1} x_i = S$$

$$R_1: \sum_{i=0}^{m-1} x_i \geq 2 \quad \text{(jugadores "pivote")}$$

$$R_2: \sum_{i=0}^{m-1} x_i \geq 3 \quad \text{(jugadores "alero")}$$

$$R_3: \sum_{i=0}^{m-1} x_i \cdot c_i \leq M$$

$$R_4: \sum_{i=0}^{m-1} x_i = 1 \quad \text{(jugadores "base")}$$

Función objetivo:

$$\text{máx} \sum_{i=0}^{m-1} (x_i \cdot t_{c_i} + x_i \cdot t_{\ell_i})$$

El archivo “suplentes.txt” contiene la información de los jugadores. Gracias a este archivo, un constructor de la clase JugadorImpl a partir de *String* y un método de la clase *Stream2,* podemos convertir la información del fichero a una lista de jugadores con la que trabajar.

```
Alex # Alero # Escolta # 1# España # 2 # 5 #1
Carlos # AleroPivot # Pivot #4 # España # 4 # 4 # 4
Jordi # Pivot # AleroPivot # 3 # España # 5 # 3 # 3
Victor # Escolta # AleroPivot #1 # España # 1 # 3 # 1
Fran # AleroPivot # Escolta # 2 # España # 2 # 5 # 2
Michael # Base # Escolta # 3 # USA # 3 # 3 # 5
Drazen # Pivot # Escolta # 1 # Croacia # 2 # 1 # 4
Emanuel # Base # Pivot # 2 # Argentina # 2 # 3 # 2
Toni # Alero # Pivot # 2 # Croacia # 2 # 5 # 2
Yao # AleroPivot # Alero # 3 # Francia # 3 # 3 # 3
Pablo # Base # Escolta # 4 # Argentina # 4 # 4 # 4
Dino # Pivot # Pivot # 2 # Croacia # 2 # 2 # 2
Lamarcus # Base # AleroPivot # 2 # USA # 2 # 2 # 2
Mark # Alero # Pivot # 1 # USA # 1 # 5 # 3
Juan # Base # Base # 3 # Argentina # 3 # 3 # 3
Homero # Pivot # AleroPivot # 4 # Argentina # 4 # 2 # 4
Chris # Base # Base # 5 # USA # 5 # 5 # 5
Joseph # AleroPivot # Escolta # 1 # Francia # 1 # 5 # 3
Zoran # Pivot # Alero # 2 # Croacia # 4 # 3 # 2
Laurent # Base # Escolta # 3 # Francia # 3 # 3 # 3
```

En la clase “ProblemaSuplentesPLI” encontramos un método que genera automáticamente el fichero LPSolve para resolver el problema. El fichero generado para nuestro ejemplo es el siguiente:

```
max: 5x0 + 1x0 + 4x1 + 4x1 + 3x2 + 3x2 + 3x3 + 1x3 + 5x4 + 2x4 + 3x5 +
5x5 + 1x6 + 4x6 + 3x7 + 2x7 + 5x8 + 2x8 + 3x9 + 3x9 + 4x10 + 4x10 +
2x11 + 2x11 + 2x12 + 2x12 + 5x13 + 3x13 + 3x14 + 3x14 + 2x15 + 4x15 +
5x16 + 5x16 + 5x17 + 3x17 + 3x18 + 2x18 + 3x19 + 3x19; // Función
objetivo

x0 + x1 + x2 + x3 + x4 + x5 + x6 + x7 + x8 + x9 + x10 + x11 + x12 +
x13 + x14 + x15 + x16 + x17 + x18 + x19 = 7; // Restricción num
suplentes

x1 + x2 + x6 + x7 + x8 + x11 + x13 + x15 + x18 >= 2; // Restricción
Pivot

x0 + x8 + x9 + x13 + x18 >= 3; // Restricción Alero

x5 + x7 + x10 + x12 + x14 + x16 + x19 = 1; // Restricción Base

1x0 + 4x1 + 3x2 + 1x3 + 2x4 + 3x5 + 1x6 + 2x7 + 2x8 + 3x9 + 4x10 +
2x11 + 2x12 + 1x13 + 3x14 + 4x15 + 5x16 + 1x17 + 2x18 + 3x19 <= 10; //
Restricción presupuesto
```

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

Para este problema existen dos soluciones: La que podemos leer en el enunciado y la que a mi me devuelve el algoritmo, ya que ambas soluciones son igual de buenas al contar con el mismo valor en la función objetivo. Esta es la segunda solución:

```
SOLUCIÓN:
0 => [(Alex), Posiciones: Alero y Escolta]
3 => [(Victor), Posiciones: Escolta y AleroPivot]
5 => [(Michael), Posiciones: Base y Escolta]
6 => [(Drazen), Posiciones: Pivot y Escolta]
8 => [(Toni), Posiciones: Alero y Pivot]
13 => [(Mark), Posiciones: Alero y Pivot]
17 => [(Joseph), Posiciones: AleroPivot y Escolta]
```

## 2 - Resolución por “Algoritmos Genéticos”

El cromosoma que yo he usado es el *BinaryChromosome,* el cual me parece a mí el más adecuado. Por otro lado, podríamos resolver el problema también con el *RangeChromosome* (siendo la multiplicidad máxima 1) o el *IndexSubListChromosome.* Para el último cromosoma mencionado, el valor del cromosoma indica el índice del jugador.

En cuanto a la solución, el algoritmo me devuelve aleatoriamente una solución de entre las dos válidas: la del enunciado y la que me devuelve el algoritmo PLI.

```
SOLUCIÓN:
0 => [(Alex), Posiciones: Alero y Escolta]
4 => [(Fran), Posiciones: AleroPivot y Escolta]
6 => [(Drazen), Posiciones: Pivot y Escolta]
7 => [(Emanuel), Posiciones: Base y Pivot]
8 => [(Toni), Posiciones: Alero y Pivot]
13 => [(Mark), Posiciones: Alero y Pivot]
17 => [(Joseph), Posiciones: AleroPivot y Escolta]
```

**ANÁLISIS Y DISEÑO DE DATOS Y ALGORITMOS. CURSO 2017/18.**
**FICHA PARA PROBLEMAS DE AG**

| **Problema** | Suplentes | **: Algoritmos Genéticos** |
|---|---|---|
| **Tipo de problema:** | ValuesInRangeProblemAG<E,S> | |
| **Tipos** | S- SolucionSupl<br>E- Jugador | |
| **Propiedades Compartidas** | L: List<Jugador><br>N: nº de jugadores<br>M: Presupuesto<br>S: nº de suplentes a contratar | $tc_i$: tiros cortos de i<br>$tl_i$: tiros largos de i<br>$P1_i$: Posición 1 de i<br>$P2_i$: Posición 2 de i |
| **Tipo de Cromosoma:** | BinaryChromosome | |
| **Decode** | d: List<Integer><br>d.size() = N.<br>$d_i$: [0,1], binaria | |

**Fitness:**

$$V - K*(var0 + var1 + var2 + var3 + var4)$$

$$V = \sum_{i=0}^{m-1} (x_i \cdot tc_i + x_i \cdot tl_i)$$

$$K = \sum_{i=0}^{m-1} (c_i)^2$$

$$var0 = \left(\sum_{i=0}^{m-1} x_i = S\right) ? 0 : 1$$

$$var1 = \left(\sum_{i=0}^{m-1} x_i \geq 2\right) ? 0 : 1$$

↳ jugadores "pivote".

$$var2 = \left(\sum_{i=0}^{m-1} x_i \geq 3\right) ? 0 : 1$$

↳ jugadores "otero".

$$var3 = \left(\sum_{i=0}^{m-1} x_i \cdot c_i \leq 4\right) ? 0 : 1$$

$$var4 = \left(\sum_{i=0}^{m-1} x_i = 1\right) ? 0 : 1$$

↳ jugadores "base".

**Solución:** `S = new SolucionSupl(m, L, S, M, fitnessFunction);`

m: Map<Integer, Jugador>. Mapa con los números de los jugadores escogidos como clave y con los jugadores asociados.