Algoritmos de genéticos

ESCUELA TÉCNICA SUPERIOR DE
INGENIERÍA INFORMÁTICA
Departamento de Lenguajes y Sistemas Informáticos
Curso 2020-2021

Contenido

1. Introducción
2. Algoritmos genéticos (AG)
3. Catálogo de tipos de cromosomas
4. Catálogo de problemas
5. Detalles de implementación
6. Conversión PLE -> AG

1. Introducción

- **Problemas de optimización**
  - Se definen por un conjunto de restricciones y una función objetivo
  - La función objetivo permite calcular un valor para cada una de las posibles soluciones
  - Se busca la solución que, cumpliendo con las restricciones del problema, minimice (o maximice) el valor de la función objetivo
- Los problemas que no son de optimización pero buscan una o varias soluciones que cumplan las restricciones del problema pueden transformarse en problemas de optimización
- En algunos casos la complejidad del algoritmo que resuelve el problema de optimización es demasiado alta
  - Por ejemplo cuando el algoritmo es exhaustivo debe encontrar todas las posibles soluciones para poder garantizar que ha encontrado el óptimo

1. Introducción

- Los **Algoritmos de Aproximación** encuentran soluciones subóptimas cercanas a la óptima pero con un menor coste de ejecución
- En este tema veremos un tipo de Algoritmos iterativos de Aproximación*: Algoritmos Genéticos*
- Son algoritmos iterativos:
  - Se diseña un Estado cuyos valores representarán las posibles soluciones del problema
  - La función objetivo se evaluará sobre el estado
  - Existirá una función que aplicada a cada estado nos indique si representa una solución válida o no, y otra función que calculará la solución asociada a los estados válidos
  - Se tendrá una condición de parada para decidir cuándo terminan
- Son algoritmos aleatorios en el sentido de que los cambios propuestos sobre el estado, en cada iteración, se generan al azar

2. Algoritmos Genéticos

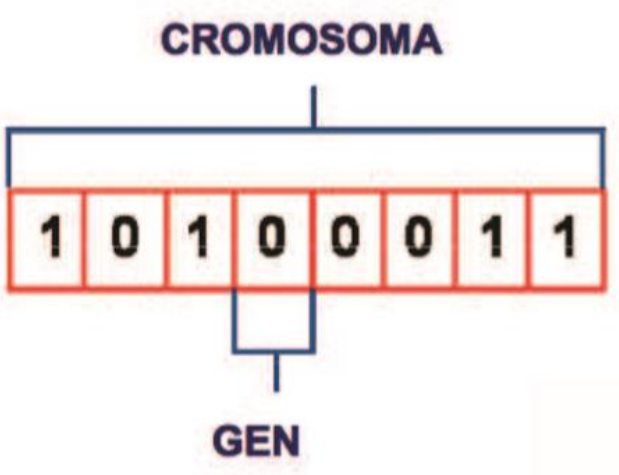
- Los ***Algoritmos Genéticos*** emulan los mecanismos de la evolución
- Estos algoritmos hacen evolucionar una población de individuos sometiéndola a cruces y mutaciones, así como también a una selección de acuerdo con algún criterio
- Llamamos generaciones a las sucesivas poblaciones que se van obteniendo haciendo evolucionar la primera de ellas
- Asumimos que queremos **maximizar** la función objetivo
- El esquema es de la forma

```
Population evolve(Population initial, StoppingCondition condition) {
        Population current = initial;
        while (!condition.isSatisfied(current)) {
                current = nextGeneration(current);
        }
        return current;
}
```

2. Algoritmos Genéticos

- La población inicial se suele inicializar de forma aleatoria
- Cada individuo tiene una representación interna que llamaremos **cromosoma**, y una medida de su fortaleza, denominada **fitness**
- El objetivo del algoritmo genético es encontrar el o los individuos que maximizan su fitness tras hacer evolucionar la población
- Un cromosoma será del tipo List<T> con algunas operaciones adicionales: cálculo del fitness, mutación y cruce
- Los elementos de la lista

  los denominaremos **genes**



2. Algoritmos Genéticos

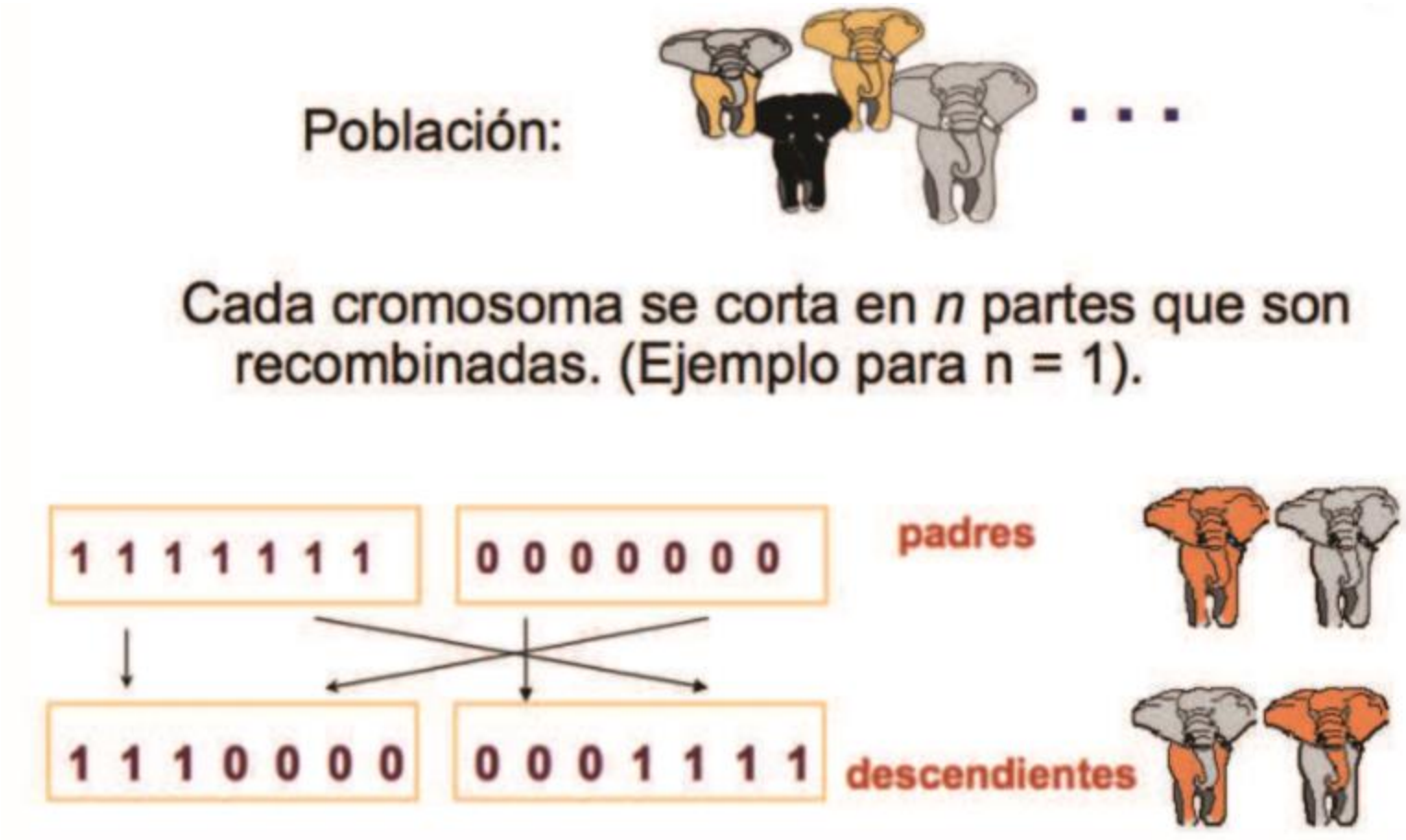
- Mecanismo para obtener la siguiente generación:

1. Se escogen los mejores individuos de una población según la tasa de elitismo escogida y se pasan a la siguiente generación
2. Se repiten los siguientes pasos hasta que la nueva generación alcanza el tamaño prefijado:
   a) Siguiendo la **Política de Selección** elegida se escogen dos cromosomas
   b) En un porcentaje establecido por la **Tasa de Cruce** se aplica el Operador de Cruce fijado
   c) En un porcentaje establecido por la **Tasa de Mutación** se aplica el Operador de Mutación fijado

- Una política de selección muy usada es la Elección por Torneo
  - Consiste en seleccionar un grupo de individuos al azar de la población y de entre ellos escoger el mejor

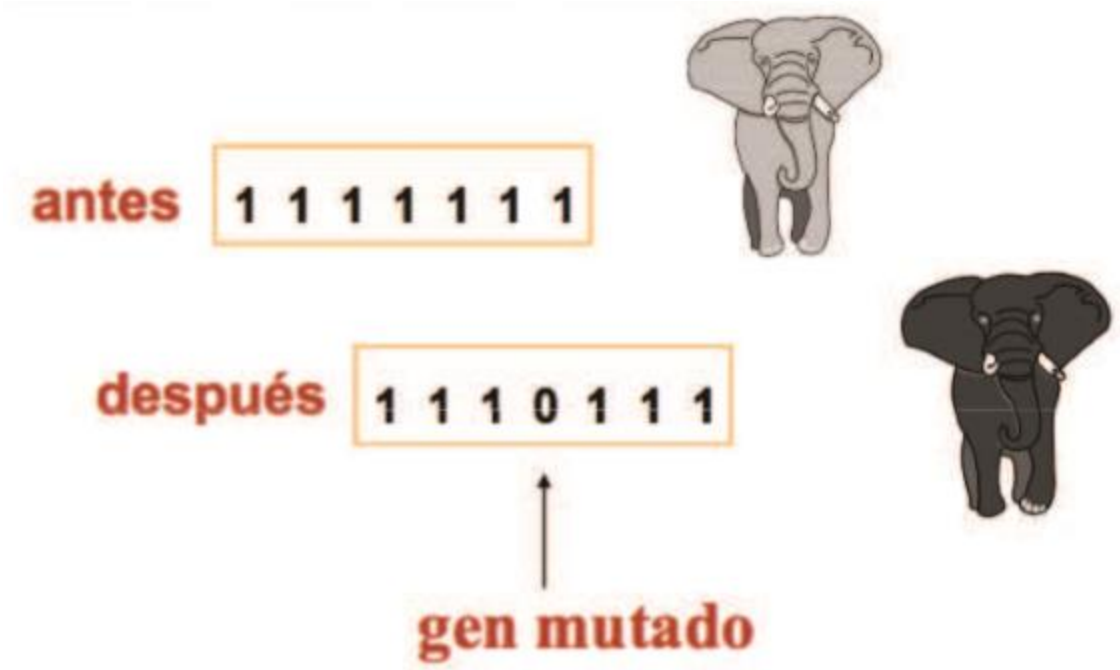
2. Algoritmos Genéticos

- Ejemplo de cruce: **OnePointCrossover**



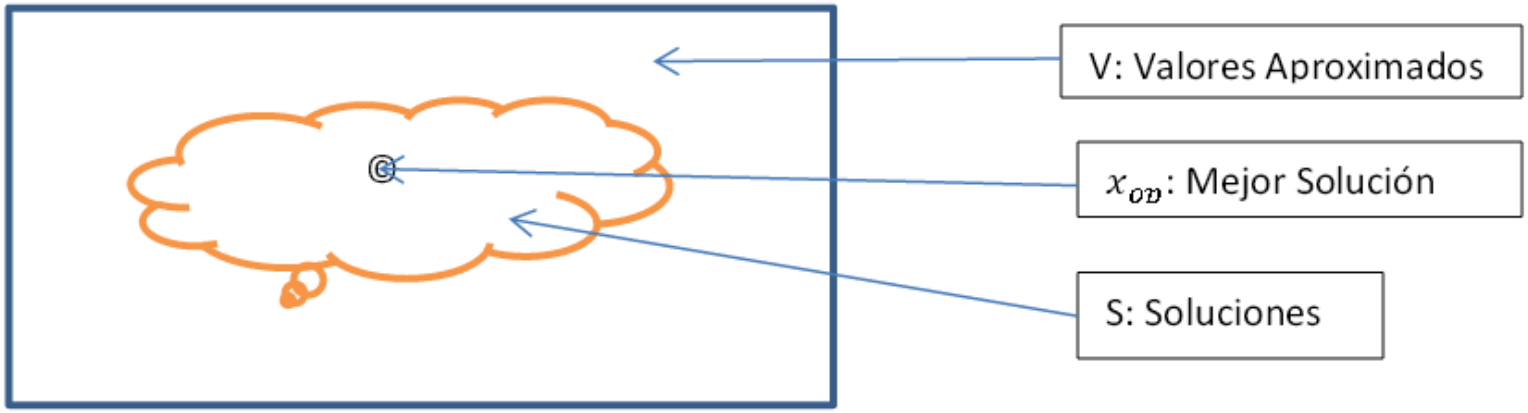
2. Algoritmos Genéticos

- Ejemplo de mutación (representación binaria)



3. Catálogo de Cromosomas

- Dependiendo del tipo de problema se escogerá el cromosoma adecuado



3. Catálogo de Cromosomas

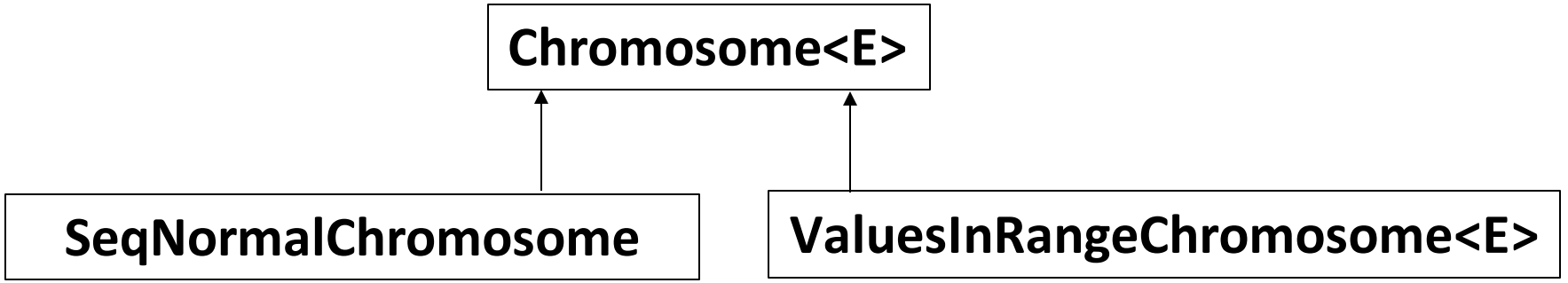
Chromosome

```
public interface Chromosome<T> {
        T decode();
        double fitness();
}
```

- Es la interfaz común a todos los cromosomas, incluye el método de decodificación y la función de fitness.
- Cualquier problema debe implementar ProblemAG o alguno de sus subtipos:
  SeqNormalProblemAG<S>, ValuesInRangeProblemAG<E,S>, ValuesInSetProblemAG<S>

```
public interface ProblemaAG {
        ChromosomeType getType();
}
```

3. Catálogo de Cromosomas



Un cromosoma cuyo valor decodificado es una lista de índices.
Adecuado para resolver problemas cuya solución es un Multiset o una lista, posiblemente con elementos repetidos, formados con elementos de un conjunto dado.

Un cromosoma cuyo valor decodificado es una lista de valores en un rango, del tamaño especificado en el problema.

3. Catálogo de Cromosomas

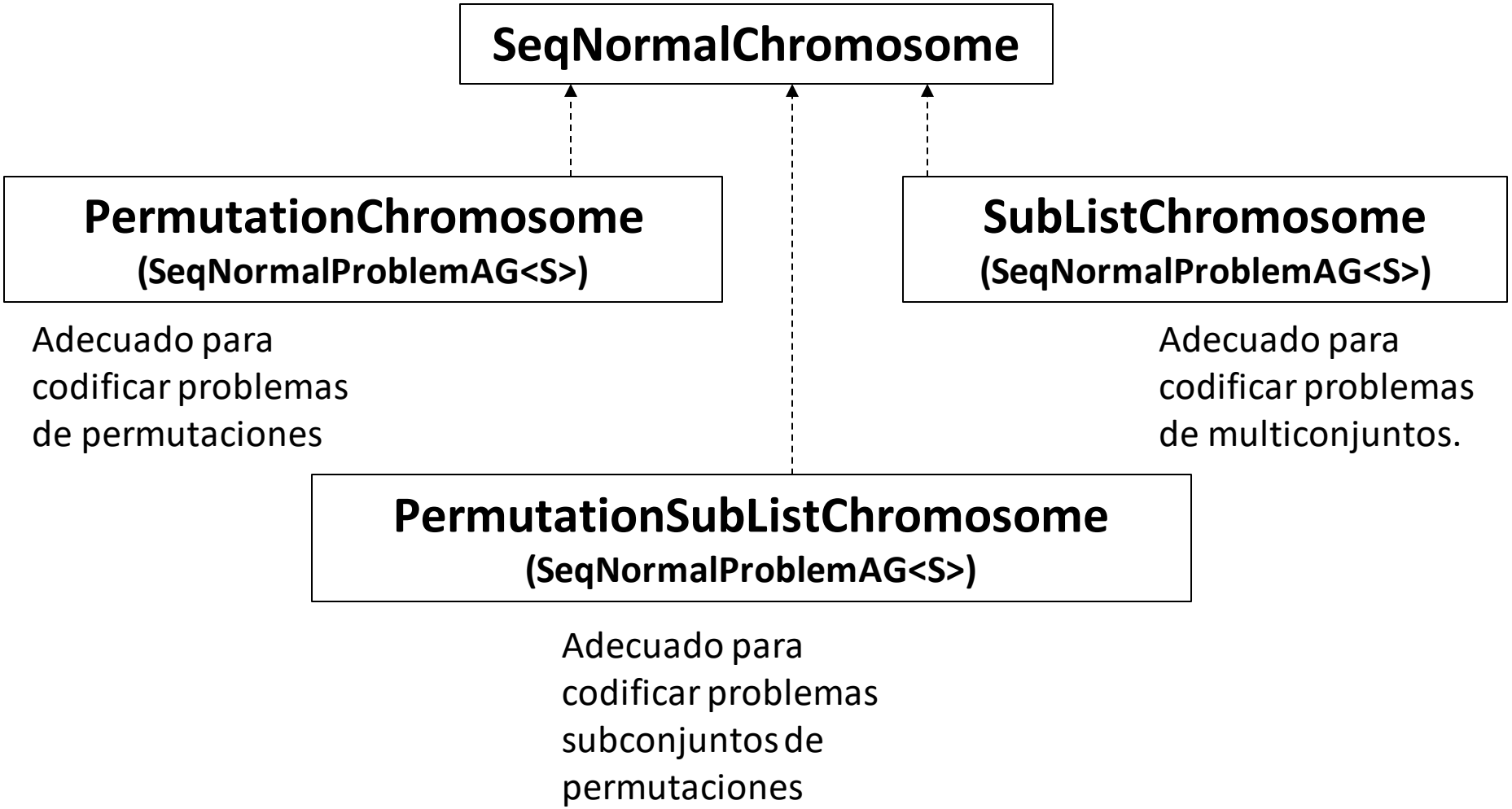
SeqNormalChromosome

- Representa una lista de valores enteros que pueden ser usados como índices en una lista de objetos dada. Si d = decode(), d[i] = índice, i: 0..n-1
- **SeqNormalProblemAG<S>**:

```
public interface SeqNormalProblemAG<S> extends ProblemAG {
        Integer getIndexNumber();
        default Integer getMaxMultiplicity(int index);
         Double fitnessFunction(SeqNormalChromosome cr);
         S getSolucion(SeqNormalChromosome cr);
        // ChromosomeType getType(); Heredado de ProblemaAG
}
```

- La secuencia normal se construye con el conjunto de n objetos distintos repetidos tantas veces como multiplicidad máxima tenga asociado cada uno

3. Catálogo de Cromosomas



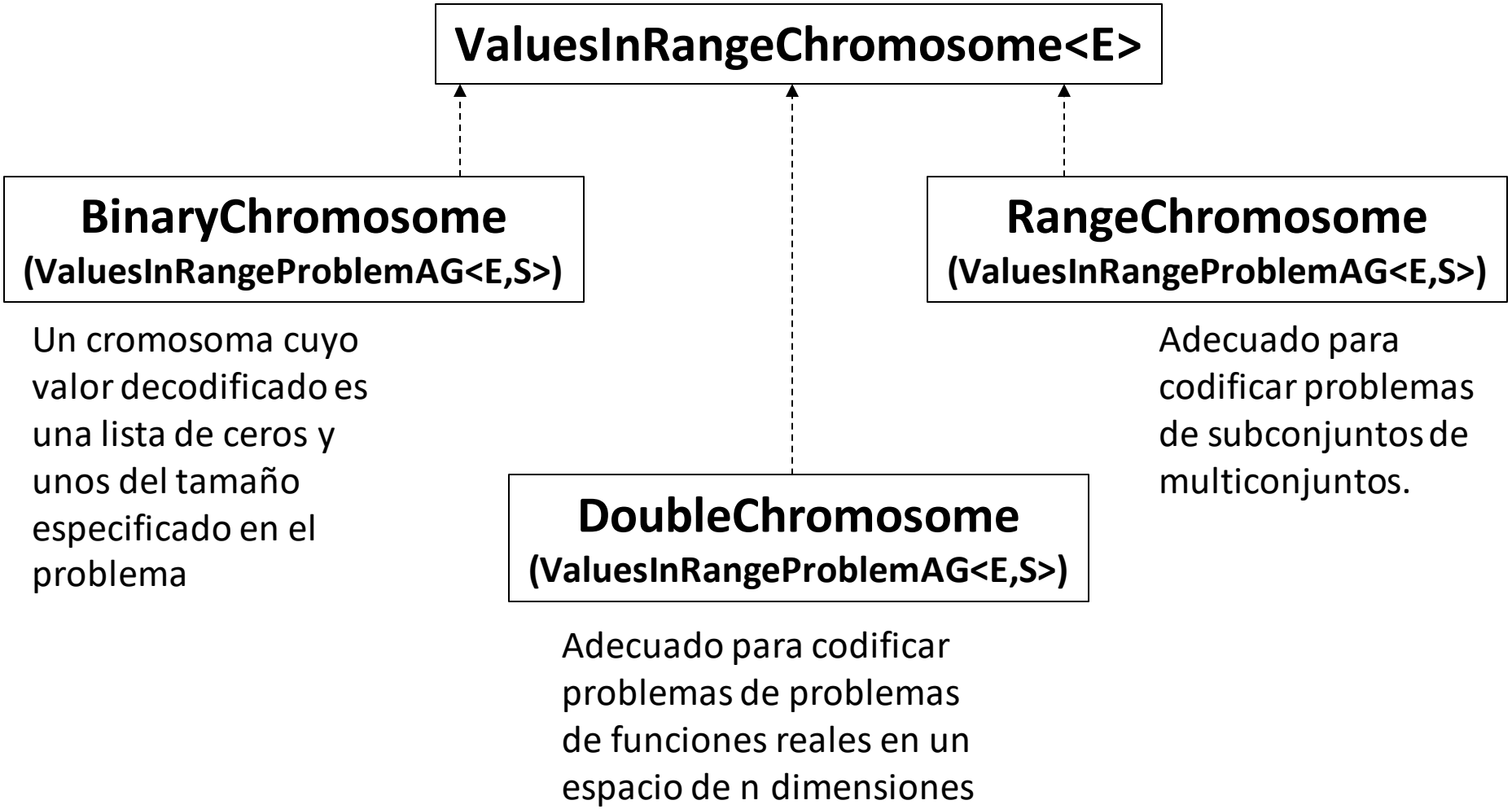
3. Catálogo de Cromosomas

ValuesInRangeChromosome<E>

- Un cromosoma cuyo valor decodificado es una lista de valores en un rango, del tamaño especificado en el problema.
- **ValuesInRangeProblemAG<E,S>**:
  - E – El tipo de los elementos del cromosoma. El cromosoma es del tipo List<E>
  - S – El tipo de la solución del problema

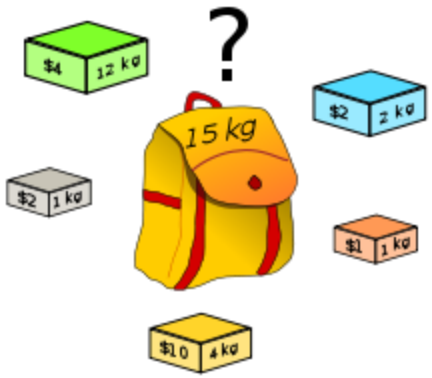
```
public interface ValuesInRangeProblemAG<E,S> extends ProblemAG {
        Integer getCellsNumber();
        E getMax(Integer i);
        E getMin(Integer i);
      Double fitnessFunction(ValuesInRangeChromosome<E> cr);
      S getSolucion(ValuesInRangeChromosome<E> cr);
      // ChromosomeType getType(); Heredado de ProblemaAG
}
```

3 Catálogo de Cromosomas



4. Catálogo de problemas

- **Problema de la Mochila**:



$$\max \sum_{i=0}^{n-1} x_i v_i$$

$$\sum_{i=0}^{n-1} x_i w_i \leq C$$

$$x_i \leq m_i, \quad i \in [0, n-1]$$

$$int \quad x_i, \quad i \in [0, n-1]$$

Cromosoma BinaryChromosome

$$0 \leq x_i = d[i] \leq 1$$

Cromosoma RangeChromosome

$$0 \leq x_i = d[i] \leq m_i$$

$$f = \sum_{i=0}^{n-1} d[i] v_i - K \; dle\left(\sum_{i=0}^{n-1} d[i] w_i - C\right)$$

$$dle(x) = \begin{cases} 0, & x \leq 0 \\ x^2, & x > 0 \end{cases}$$

Penaliza si es mayor que cero

4. Catálogo de problemas

- **Problema de la Asignación**:

	T1	T2	T3	T4
A1	14	5	8	7
A2	2	12	6	5
A3	7	8	3	9
A4	2	4	6	10

$$\min \sum_{i=0,j=0}^{n-1,n-1} x_{ij}\, c_{ij}$$

$$\sum_{j=0}^{n-1} x_{ij} = 1, \quad i \in [0, n-1]$$

$$\sum_{i=0}^{n-1} x_{ij} = 1, \quad j \in [0, n-1]$$

$$bin\, x_{ij}, \quad i \in [0, n-1], \quad j \in [0, n-1]$$

Cromosoma PermutationChromosome

$$f = -\sum_{i=0}^{n-1} c_{id[i]}$$

4. Catálogo de Problemas

- Problema de los Anuncios
  - Un canal de televisión quiere obtener el máximo rendimiento (en euros) de la secuencia de anuncios que aparecerá en un corte de publicidad de un máximo de T segundos
  - El precio final del anuncio (p) dependerá de la posición que éste ocupe dentro de la secuencia

$$p(i) = \frac{b * t(i)}{pos(i)} + c$$

  - b y c son constantes

- Ejemplo:

Cliente:	1	2	3	4	5	6	7
Tiempo Anuncio (segundos):	5	25	25	5	15	10	7

4. Catálogo de problemas

- **Problema de los Anuncios**:

$$\max \sum_{i=0}^{s-1} \left(\frac{b\, t(d[i])}{i+1} + c\right)$$

$$s \leq n$$

$$\sum_{i=0}^{s-1} t(d[i]) \leq T$$

Cromosoma PermutationSubListChromosome

$$f = \sum_{i=0}^{s-1} \left(\frac{b\, t(d[i])}{i+1} + c\right) - K\, dge\left(T - \sum_{i=0}^{s-1} t(d[i])\right)$$

$$dge(x) = \begin{cases} 0, & x \geq 0 \\ x^2, & x < 0 \end{cases}$$

Penaliza si es menor que cero

$$f = \sum_{i=0}^{s-1} \left(\frac{b\, t(d[i])}{i+1} + c\right)$$

Si se trunca el cromosoma.

5. Detalles de implementación

La implementación es una adaptación del software del proyecto Apache

El esquema del Algoritmo Genético es de la forma:

```
public Population evolve(Population initial, StoppingCondition condition) {
        Population current = initial;
        while (!condition.isSatisfied(current)) {
                current = nextGeneration(current);
        }
        return current;
}
```

Ejemplos:

- Problema de la Mochila (Range)
- Problema de la Asignación (Permutation)

5. Detalles de implementación

La clase encargada de resolver el algoritmo AG es AlgoritmoAG

```
public abstract class AlgoritmoAG<C> {
        public static int POPULATION_SIZE  = 30;
        public static double ELITISM_RATE  = 0.2;
        public static double CROSSOVER_RATE = 0.8;
        public static double MUTATION_RATE  = 0.6;
        public static long INITIAL_TIME;
        private ChromosomeType tipo;
        private CrossoverPolicy crossOverPolicy;
        private MutationPolicy mutationPolicy;
        private SelectionPolicy selectionPolicy;
        protected StoppingCondition stopCond;
        public static List<Chromosome> bestChromosomes;
        protected Population initialPopulation;
        protected Chromosome bestFinal;
        protected Population finalPopulation;

        public AlgoritmoAG(ProblemaAG problema) {
                super();
                this.tipo = problema.getType();
                …
        }
```

5. Detalles de implementación

```
public void ejecuta() {
        INITIAL_TIME  = System.currentTimeMillis();
        this.initialPopulation = randomPopulation();
        Preconditions.checkNotNull(this.initialPopulation);
        GeneticAlgorithm ga = new GeneticAlgorithm(
                                crossOverPolicy,
                                CROSSOVER_RATE,
                                mutationPolicy,
                                MUTATION_RATE,
                                selectionPolicy);
        this.finalPopulation = ga.evolve(this.initialPopulation, this.stopCond);
        Preconditions.checkNotNull(this.finalPopulation);
        this.bestFinal = this.finalPopulation.getFittestChromosome();
}

public Population getInitialPopulation() { return initialPopulation; }
public C getBestChromosome() { return (C)bestFinal; }
public List<C> getBestChormosomes() { … }
public Population getFinalPopulation() { return finalPopulation; }

public static <C> AlgoritmoAG<C> create(ProblemAG p) {
        return new AlgoritmoAG<C>(p);
}
}
```

5 Detalles de implementación

- Para la condición de parada las opciones posibles son:
  - Tiempo transcurrido. Se acaba cuando pase el tiempo indicado.
  - Número de generaciones máximo.
  - Número de soluciones distintas encontradas.
  - Alguna combinación de las anteriores.

5. Detalles de implementación

- Parámetros de configuración del algoritmo genético:
  - DIMENSION: Dimensión del cromosoma.
  - POPULATION_SIZE: Tamaño de la población.
  - NUM_GENERATIONS: Número de generaciones.
  - ELITISM_RATE: Tasa de elitismo. El porcentaje especificado de los mejores cromosomas pasa a la siguiente generación sin cambio. Valor usual 0.2.
  - CROSSOVER_RATE: Tasa de cruce. Indica con qué frecuencia se va a realizar el cruce. Si no hay un cruce, la descendencia es copia exacta de los padres. Si hay un cruce, la descendencia está hecha de partes de los cromosomas de los padres. Valores usuales entre 0.8 y 0.95.
  - MUTATION_RATE: Tasa de mutación. Indica con qué frecuencia serán mutados cada uno de los cromosomas. Si no hay mutación, la descendencia se toma después de cruce sin ningún cambio. La mutación se hace para evitar que se caiga en un máximo local. Valores usuales entre 0.5 y 1.
  - TOURNAMENT_ARITY: Número de participantes en el torneo para elegir los cromosomas que participarán en el cruce y mutación. Valor usual 2.

6. Conversión PLE -> AG

- Los problemas de optimización que buscan minimizar una función objetivo se suelen representar como:

$$\min_{x \in \Omega} f(x)$$

- Donde x es un vector de n variables que deben tomar valores dentro de un dominio. Los valores de x que están dentro del dominio se dice que son valores válidos
- Los dominios se pueden representar como restricciones de desigualdad, igualdad, o rango:

$$g(x) \leq 0$$

$$h(x) = 0$$

$$a \leq x \leq b$$

  - Donde a y b son vectores de valores

6. Conversión PLE -> AG

- Las restricciones de un problema pueden ser introducidas en la función objetivo. Algunas equivalencias son:

$$\begin{aligned} &\min_x f(x) \\ &g_i(x) \le 0, \;\; i = 1, \dots, r \\ &h_j(x) = 0, \;\; j = 1, \dots, s \\ &a \le x \le b \end{aligned} \quad \equiv \quad \max_{\substack{x \\ a \le x \le b}} \left(-f(x) - K\left(\sum_{i=1}^{r} (c(g_i(x)))^2 - \sum_{j=1}^{s} (h_j(x))^2\right)\right)$$

$$\begin{aligned} &\max_x f(x) \\ &g_i(x) \le 0, \;\; i = 1, \dots, r \\ &h_j(x) = 0, \;\; j = 1, \dots, s \\ &a \le x \le b \end{aligned} \quad \equiv \quad \max_{\substack{x \\ a \le x \le b}} \left(f(x) - K\left(\sum_{i=1}^{r} (c(g_i(x)))^2 - \sum_{j=1}^{s} (h_j(x))^2\right)\right)$$

$$c(z) = \max(0, z) = \begin{cases} 0, & z \le 0 \\ z, & z > 0 \end{cases}$$

6. Conversión PLE -> AG

- Los problemas de maximización pueden transformarse a problemas de minimización de la forma:

$$\max_{x \in \Omega} f(x) = \min_{x \in \Omega} -f(x)$$

- En los problemas multiobjetivo hay que optimizar simultáneamente varios objetivos:

$$\min_{x \in \Omega} (f_1(x), \ldots, f_m(x))$$

- Puede ser reducido a un problema uniobjetivo de la forma:

$$\min_{x \in \Omega} \sum_{i=1}^{m} \omega_i f_i(x)$$

$$\sum_{i=1}^{m} \omega_i = 1$$

Algoritmos de genéticos

ESCUELA TÉCNICA SUPERIOR DE
INGENIERÍA INFORMÁTICA
Departamento de Lenguajes y Sistemas Informáticos
Curso 2020-2021