

# Metaheurísticas

## Seminario 3. Problemas de optimización con técnicas basadas en poblaciones

---

1. Estructura de un Algoritmo Genético/Memético y Aspectos de Implementación
2. Problemas de Optimización con Algoritmos Genéticos y Meméticos
  - Máxima Diversidad
  - Aprendizaje de Pesos en Características

# Estructura de un Algoritmo Genético

## Procedimiento Algoritmo Genético

Inicio (1)

$t = 0;$

inicializar  $P(0)$  de forma aleatoria;

evaluar  $P(t)$ ;

Mientras (no se cumpla la condición de parada) hacer

Inicio(2)

$t = t + 1$

seleccionar  $P'$  desde  $P(t-1)$

recombinar  $P'$

mutar  $P'$  El operador de mutación no tiene tanta importancia

reemplazar  $P(t)$  a partir de  $P(t-1)$  y  $P'$

evaluar  $P(t)$

Final(2)

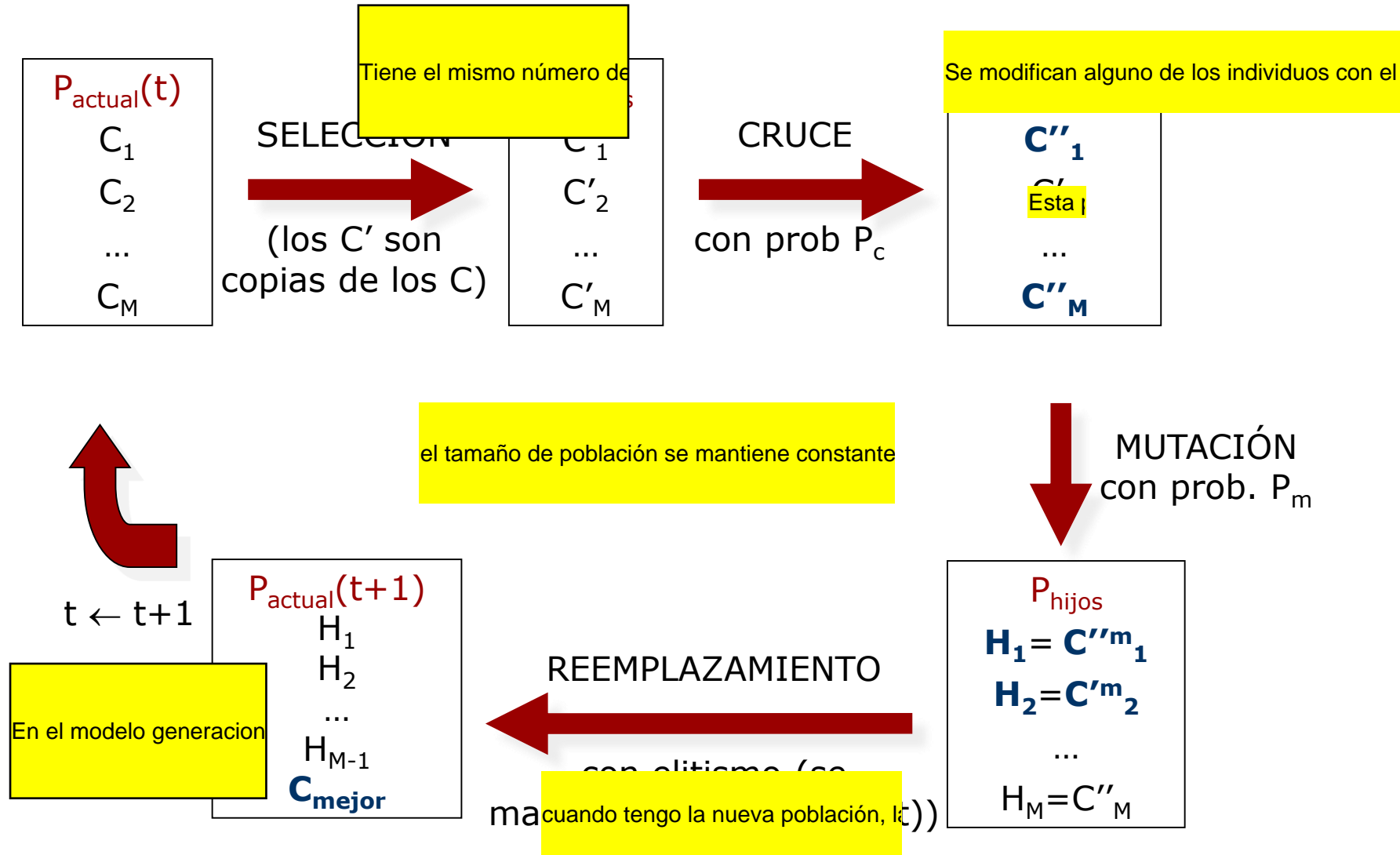
Final(1)

Tenemos varias soluciones

Tenemos dos operadores, el de selección y el de cruce

Uno de los modelos que vemos

# Modelo Generacional



# Modelo Generacional:

## Aspectos de Implementación

---

- ✓ Lo mas costoso en tiempo de ejecución de un Algoritmo Genético es la generación de números aleatorios para:
  - ✓ Aplicar el mecanismo de selección
  - ✓ Emparejar las parejas de padres para el cruce
  - ✓ Decidir si una pareja de padres cruza o no de acuerdo a  $P_c$
  - ✓ **Decidir si cada gen muta o no de acuerdo a  $P_m$**
- ✓ Se pueden diseñar **impr** En la selección no se podrá reducir el coste de l **es que reduzcan** en gran medida la cantidad de números aleatorios necesaria:
  - ✓ Emparejar las parejas para el cruce: Como el mecanismo de selección ya tiene una **los individuos ya están desorden**a, se aplica siempre un emparejamiento fijo: el primero con el segundo, el tercero con el cuarto, etc.

# Modelo Generacional:

## Aspectos de Implementación

---

- ✓ Decidir si una pareja de padres cruza: En vez de generar un aleatorio  $u$  en  $[0,1]$  para cada pareja y cruzarla si  $u \leq P_c$ , se estima a priori (al principio del algoritmo) el número de cruces a hacer en cada generación (**esperanza matemática**):

$$N^o \text{ esperado cruces} = P_c \cdot M/2$$

- ✓ Por ejemplo, con una población de 60 cromosomas (30 parejas) y una  $P_c$  de 0.6, cruzarán  $0,6 \cdot 30 = 18$  parejas
- ✓ De nuevo, consideramos la aleatoriedad que ya aplica el mecanismo de selección y cruzamos siempre las  $N^o \text{ esperado cruces}$  primeras parejas de la población intermedia

# Modelo Generacional:

## Aspectos de Implementación

---

- ✓ Decidir si cada gen muta: El problema es similar al del cruce, pero mucho mas acusado
- ✓ Normalmente, tanto el tamaño de población  $M$  como el de los cromosomas  $n$  es grande. Por tanto, el número de genes de la población,  $M \cdot n$ , es muy grande
- ✓ La  $P_m$ , definida a nivel de gen, suele ser muy baja (p.e.  $P_m = 0.01$ ). Eso provoca que se generen muchos números aleatorios para finalmente realizar muy pocas mutaciones
- ✓ Por ejemplo, con una población de 60 cromosomas de 100 genes cada uno tenemos 6000 genes de los cuales mutarían unos 60 (*Nº esperado mutaciones* =  $P_m \cdot n^\circ \text{ genes población}$ , *esperanza matemática*)
- ✓ Generar 6000 números aleatorios en cada generación para hacer sólo 60 mutaciones (en media) es un gasto inútil. Para evitarlo, haremos siempre exactamente  $n$  números aleatorios. Ejemplo: lanzo un número aleatorio entre 0 y 6000 para saber que individuo mutar, y otra para saber que gen mutar.

# Modelo Generacional:

## Aspectos de Implementación

---

- ✓ Aparte de hacer un número fijo de mutaciones, hay que decidir cuáles son los genes que mutan
- ✓ Normalmente, eso se hace también generando números aleatorios, en concreto dos, un entero en  $\{1, \dots, M\}$  para escoger el cromosoma y otro en  $\{1, \dots, n\}$  para el gen
- ✓ Existen también mecanismos más avanzados que permiten escoger el gen a mutar generando un único número real en  $[0,1]$  y haciendo unas operaciones matemáticas (ver código entregado en prácticas)

En la mutación se puede mutar varias veces el mismo cromosoma

La mutación se hace en probabilidades pequeñas porque puede provocar la pérdida de la diversidad genética

No es bueno que toda la población sea muy parecida

# Aspectos de Diseño

Se añade el algoritmo local, porque en los genéticos es difícil incluir información

- Una decisión fundamental en el diseño de un Algoritmo Memético (AM) es la definición del equilibrio entre:
  - la exploración desarrollada por el algoritmo de búsqueda global (el Algoritmo Genético (AG)) y
  - la explotación desarrollada por el algoritmo de búsqueda local (BL)
- La especificación de este **equilibrio entre exploración y explotación** se basa principalmente en dos decisiones:
  1. ¿Cuándo se aplica el optimizador local
    - En cada **Esto realizando muchos esfuerzo a la BL,**
    - cada **El profesor suele hacer 50/50 ó 60/40** iteracionesy sobre **sobre que individuos de la** **que agentes?**
  - Sólo sobre el mejor individuo de la población en la generación actual o
  - sobre un subconjunto de individuos escogidos de forma fija (los  $m$  mejores de la población) o variable (de acuerdo a una probabilidad de aplicación  $p_{LS}$ )



# Aspectos de Diseño de los Algoritmos Meméticos

2. ¿Sobre qué agentes se aplica (más o menos individuos elegidos) y con qué intensidad (proporción de iteraciones reemplazadas por la BL)?

- AMs baja intensidad (alta frecuencia de aplicación de la BL/pocas iteraciones)
- AMs alta intensidad (baja frecuencia de la BL/muchas iteraciones)

Creo que este es

Lo suyo es aplicar mayor intensidad a la r

El profesor plantea, si yo aplico dos veces BL con 50 iteraciones es como hacer una única BL con 100 iteraciones

Ver modelo estacionario en las transparencias de teoría. Creo que en vez de sustituir por completo los hijos a los padres

Cambiar de un modelo es fácil. El profesor nos recomienda en

# Problema de la Máxima Diversidad (MDP)

## ■ Problema de la Máxima Diversidad MaxSum, *MDP*:

*Seleccionar un subconjunto  $M$  de  $m$  elementos ( $|M|=m$ ) de un conjunto inicial  $N$  de  $n$  elementos de forma que se maximice la diversidad entre los elementos escogidos calculada como la suma de las distancias entre cada par de esos elementos*

Ahora una solución en vez de ser una lista de enteros, va a ser una lista de 0 y 1 que nos ir

$$\text{Maximizar } z_{MS}(x) = \sum_{i=1} \sum_{j=i+1} d_{ij} x_i x_j$$

$$\text{Sujeto a } \sum_{i=1}^n x_i = m$$

$$x_i = \{0, 1\}, \quad i = 1, \dots, n.$$

donde  $x$  es el vector binario solución al problema

# Algoritmo Genético para el MDP

Katayama, Narihisa. An Evolutionary Approach for the Maximum Diversity Problem. En: Hart, Krasnogor, Smith (Eds.), Recent Advances in Memetic Algorithms, vol. 166, 2005, 31–47

- **Representación binaria:** vector binario  $Se = (x_1, \dots, x_n)$  en el que las posiciones del vector representan los elementos y su valor, 0 o 1, la no selección o selección de los mismos

Para que la solución candidata codificada sea factible tiene que verificar las restricciones

Hay que asegurarse que las soluciones cumplan

debe haber exactamente el número de unos igual al número de elementos

- **Generación de la población inicial:** aleatoria **verificando las restricciones**
- **Modelos de evolución:** 2 variantes: generacional con elitismo / estacionario con 2 hijos que compiten con los dos peores de la población
- **Mecanismo de selección:** torneo binario **binario porque el torneo es de dos**
- **Operador de mutación:** **La mutación se hace intercambiando un cero con un uno, sólo se hace de esta manera para**  
 $x_i$  por el de otro gen  $x_j$  **escogidos aleatoriamente con el valor contrario**

# Algoritmo Genético para el MDP

Da buenos resultados, pero puede dar soluciones no factibles, por lo que después del proceso de cruce hay que

## Operador de Cruce 1: Cruce uniforme (requiere reparador)

- Genera un hijo a partir de dos padres. Para generar dos hijos, lo ejecutaremos dos veces a partir de los mismos padres
- Aquellas posiciones que contengan <sup>sea 1 ó 0</sup> el mismo valor en ambos padres se mantienen en el hijo (para preservar las selecciones prometedoras)
- Las selecciones restantes se seleccionan aleatoriamente de un padre o del otro. Ejemplo con  $n=9$  y  $m=5$ :

Padre<sub>1</sub> = (0 1 1 0 0 0 1 1 1)

Padre<sub>2</sub> = (1 0 1 1 1 0 1 0 0)

Hijo' = (\* \* 1 \* \* 0 1 \* \*)

Hijo = (1 1 1 0 1 0 1 1 0)

**¡OJO! Es una solución no factible**

# Algoritmo Genético para el MDP

Los algoritmos genéticos tardan mucho más que los

operadores para convertir las soluciones no factibles

## Operador de reparación

$S_1$ : conjunto de elementos seleccionados en  $x$ ;  $S_0$ : conjunto de elementos NO seleccionados en  $x$ ;  $g_j$ : contribución del elemento  $j$  al coste de la solución  $x$

```
procedure Repair( $x, g$ )
begin
1  calculate a violation  $v := m - |S_1|$ ;
2  if  $v = 0$  then return  $x$ ;
3  else if  $v < 0$  then
4    repeat
5      find  $j$  with  $g_j = \max_{j \in S_1} g_j$ ;
6       $x_j := 1 - x_j$ ,  $S_1 := S_1 \setminus \{j\}$ , and update gains  $g$ ;
7      until  $\sum_{i=1}^n x_i = m$ ;
8      return  $x$ ;
9  else
10   repeat
11     find  $j$  with  $g_j = \max_{j \in S_0} g_j$ ;
12      $x_j := 1 - x_j$ ,  $S_0 := S_0 \setminus \{j\}$ , and update gains  $g$ ;
13     until  $\sum_{i=1}^n x_i = m$ ;
14     return  $x$ ;
15   endif
end;
```

**Se chequea la factibilidad de  $x$ . Si selecciona  $m$  elementos, no se hace nada**

**Si solo faltan algunos elementos, se eliminan los de mayor contribución hasta que  $x$  sea factible**

**Si faltan elementos, se van añadiendo los de mayor contribución hasta que  $x$  sea factible**

Por lo que he entendido si elego un padre con r

# Algoritmo Genético para

Para obtener el fitness de una solución, se puede tr

## Operador de cruce basado en

es más aleatorio pero siempre da solu

Este operador será más rápido que e

- Aquellas posiciones que contengan el mismo valor en ambos padres se mantienen en el hijo (para preservar las selecciones prometedoras)
- Las asignaciones restantes se toman de un padre (da igual de cual) y se asignan en un orden aleatorio distinto para completar cada hijo. Ejemplo ( $n=9$  y  $m=5$ ):

Padre<sub>1</sub> = (0 1 1 0 0 0 1 1 1)

Padre<sub>2</sub> = (1 0 1 1 1 0 1 0 0)

Hijo' = (\* \* 1 \* \* 0 1 \* \*)

Restos Padre<sub>1</sub>: {0, 1, 0, 0, 1, 1} → Orden aleatorio<sub>1</sub>: {1, 1, 0, 0, 1, 0}

Orden aleatorio<sub>2</sub>: {0, 1, 0, 1, 0, 1}

Hijo<sub>1</sub> = (1 1 1 0 0 0 1 1 0)

Hijo<sub>2</sub> = (0 1 1 0 1 0 1 0 1)

Se deben generar dos hijos, por lo que se cogen c

**Genera hijos factibles si los dos padres son factibles. Es más disruptivo que el otro, comparte menos información de los padres, puede ser más complicado que converja**

# Problema del agrupamiento con restricciones (PAR)

---

**El Problema del Agrupamiento con Restricciones (PAR)** bajo **restricciones débiles** consiste en minimizar una agregación de *infeasibility* (número de restricciones incumplidas) y  $\bar{c}$  (la desviación general) permitiendo *infeasibility*  $> 0$

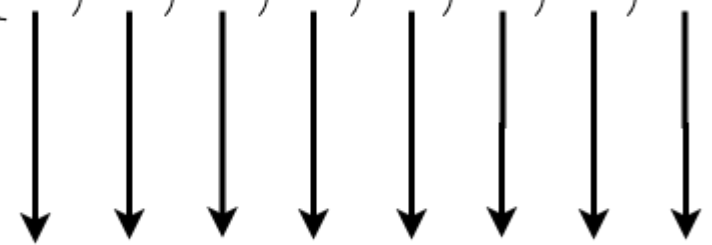
Para abordar este problema mediante AGs debemos definir la representación del mismo, así como los operadores propios de esta metaheurística

# Algoritmo Genético para el PAR:

## Representación de la solución

---

Emplearemos el modelo de representación basado en etiquetas que ya utilizamos para la BL de la primera práctica. Cada individuo de la población consiste en un vector de  $n$  posiciones en correspondencia con las  $n$  instancias del conjunto de datos  $X$ . En cada posición se almacena la etiqueta de la instancia asociada a dicha posición:

$$C = \{ \overset{x_0}{1}, \overset{x_1}{1}, \overset{x_2}{1}, \overset{x_3}{1}, \overset{x_4}{2}, \overset{x_5}{2}, \overset{x_6}{2}, \overset{x_7}{2} \}$$

$$S = [\overset{s_0}{1}, \overset{s_1}{1}, \overset{s_2}{1}, \overset{s_3}{1}, \overset{s_4}{2}, \overset{s_5}{2}, \overset{s_6}{2}, \overset{s_7}{2}]$$



# AG para el PAR: Función Objetivo

---

De nuevo emplearemos la misma función objetivo que en la práctica 1. Con el mismo método para el cálculo del parámetro  $\lambda$ .

$$f = \overline{C} + (infeasibility * \lambda)$$

Desviación General

Número De Restricciones Incumplidas

# AG para el PAR: Aspectos Generales

---

- **Generación de la población inicial:** aleatoria, para cada cromosoma generamos  $n$  valores en el intervalo  $\{1, \dots, k\}$  de manera uniforme, **verificando las restricciones**
- **Modelos de evolución:** 2 variantes: generacional con elitismo/estacionario con 2 hijos que compiten con los dos peores de la población
- **Mecanismo de selección:** torneo binario
- **Operador de cruce:** 2 variantes, que generan un único hijo por cada pareja de padres. Se aplican 2 veces sobre cada pareja de padres para obtener 2 descendientes

Ambos **pueden generar soluciones no factibles** por dejar algún clúster vacío. En ese caso, se aplicaría una reparación, moviendo un elemento escogido aleatoriamente a dicho cluster

# AG para el PAR: Operador de cruce uniforme

---

El operador de cruce uniforme produce un nuevo individuo (hijo) en base a dos individuos dados (padres) combinando de manera uniforme las características de los dos padres

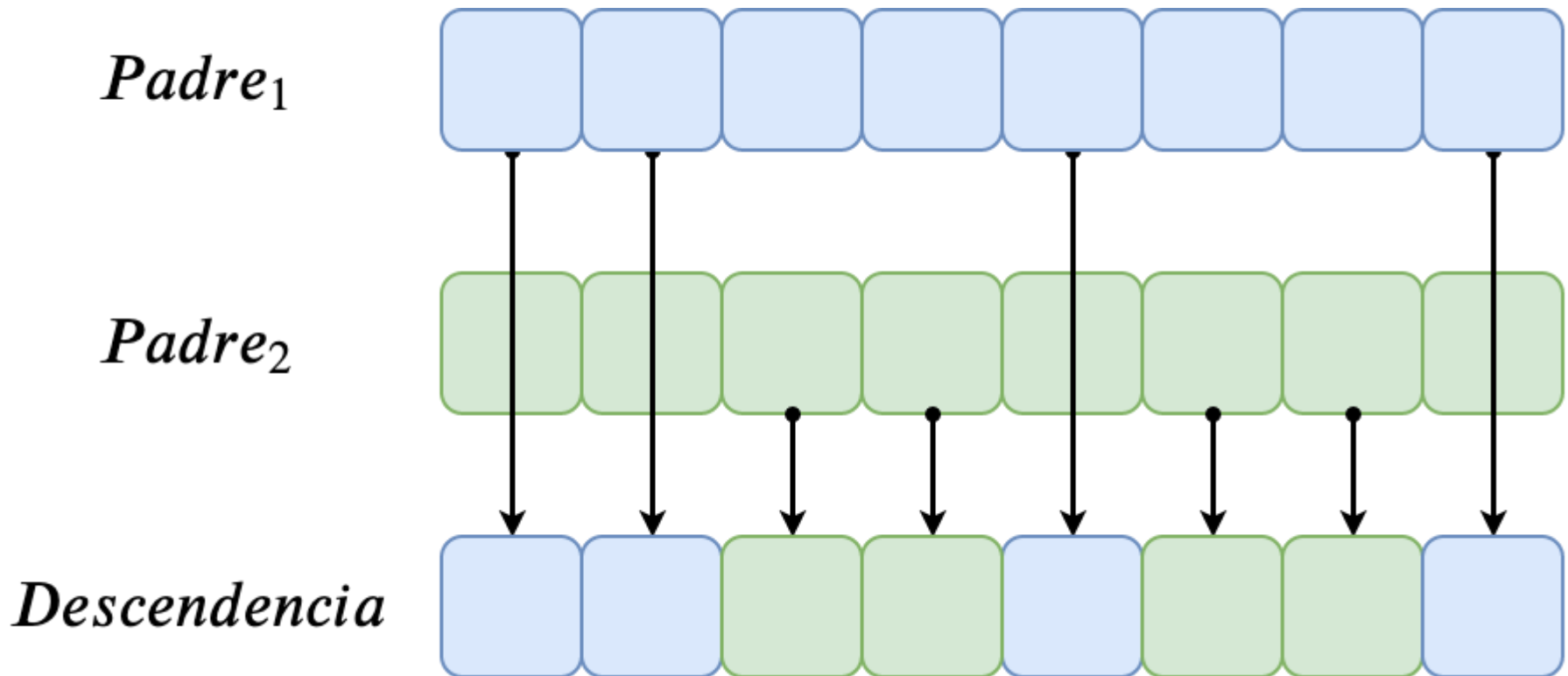
Para ello se seleccionan aleatoriamente la mitad de los genes de un padre y la otra mitad de otro padre

**Procedimiento:** generamos  $n/2$  números aleatorios distintos en el rango  $\{0, \dots, n - 1\}$ . Seleccionamos uno de los padres y copiamos en la descendencia los genes cuyo índice coincide con los números generados. Los genes que quedan por asignar en la descendencia se copian del segundo padre

Para generar dos hijos, lo ejecutaremos dos veces a partir de los mismos padres

# AG para el PAR: Operador de cruce uniforme

- Ejemplo para  $n = 8$ . Obtenemos la lista de números aleatorios  $\{0, 1, 4, 7\}$ . Tomamos el primer progenitor como  $Padre_1$ :



# AG PAR: Operador de cruce por segmento fijo

---

El operador de cruce por segmento fijo produce un nuevo individuo (hijo) en base a dos individuos dados (padres) seleccionando de uno de ellos un segmento continuo de características y copiándolo sin modificación a la descendencia

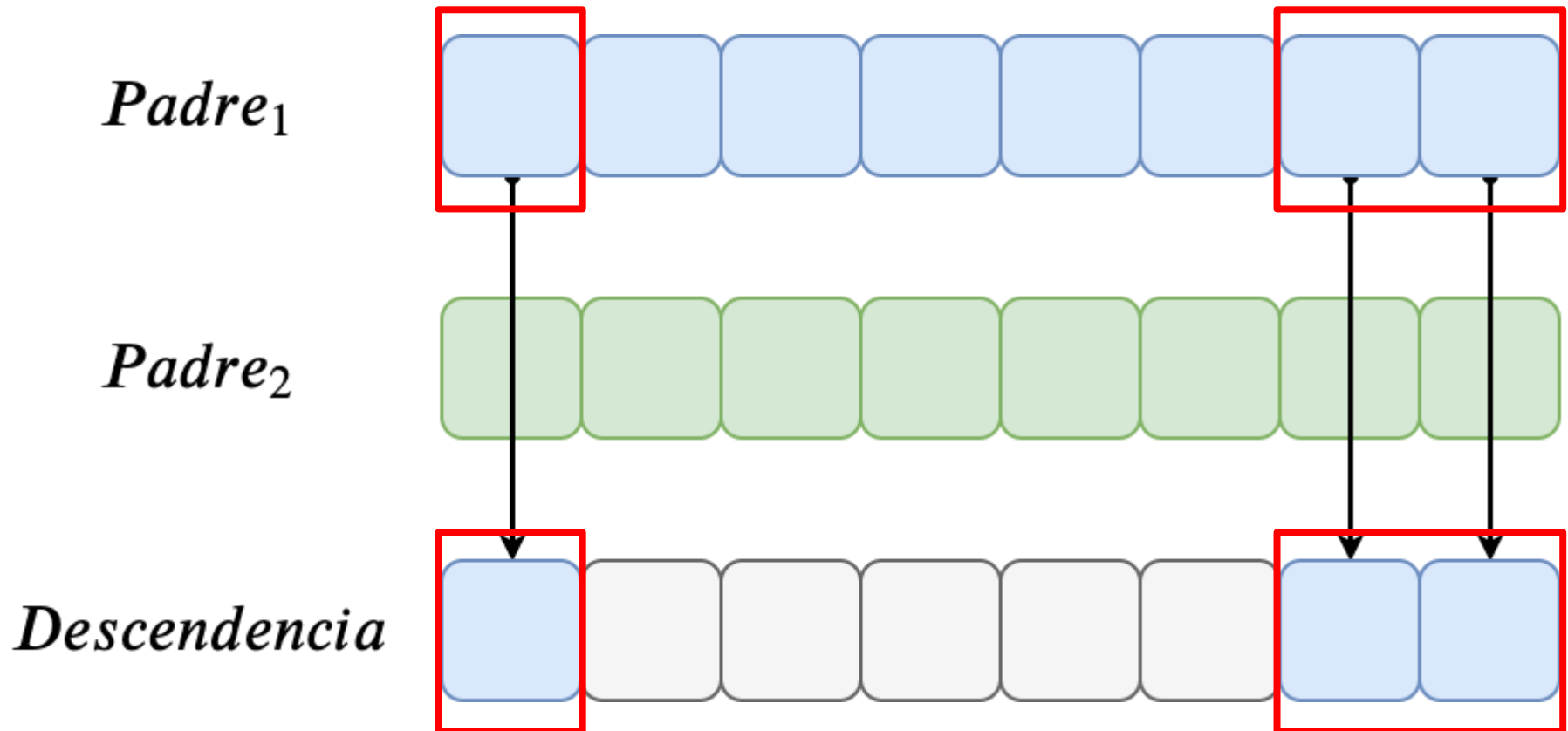
Los genes que quedan por asignar en la descendencia combinan de manera uniforme características de los dos padres

El tamaño de segmento  $v$  se genera aleatoriamente para cada aplicación del cruce para dar diversidad

En cada cruce se generan dos números aleatorios  $r$  (inicio del segmento) y  $v$  (tamaño del segmento) en el rango  $\{0, \dots, n - 1\}$ . Seleccionamos un padre y copiamos a la descendencia los genes con los índices en el intervalo  $\{r, ((r + v) \bmod n) - 1\}$ . El resto de genes de la descendencia se determinan igual que en el cruce uniforme

# AG PAR: Operador de cruce por segmento fijo

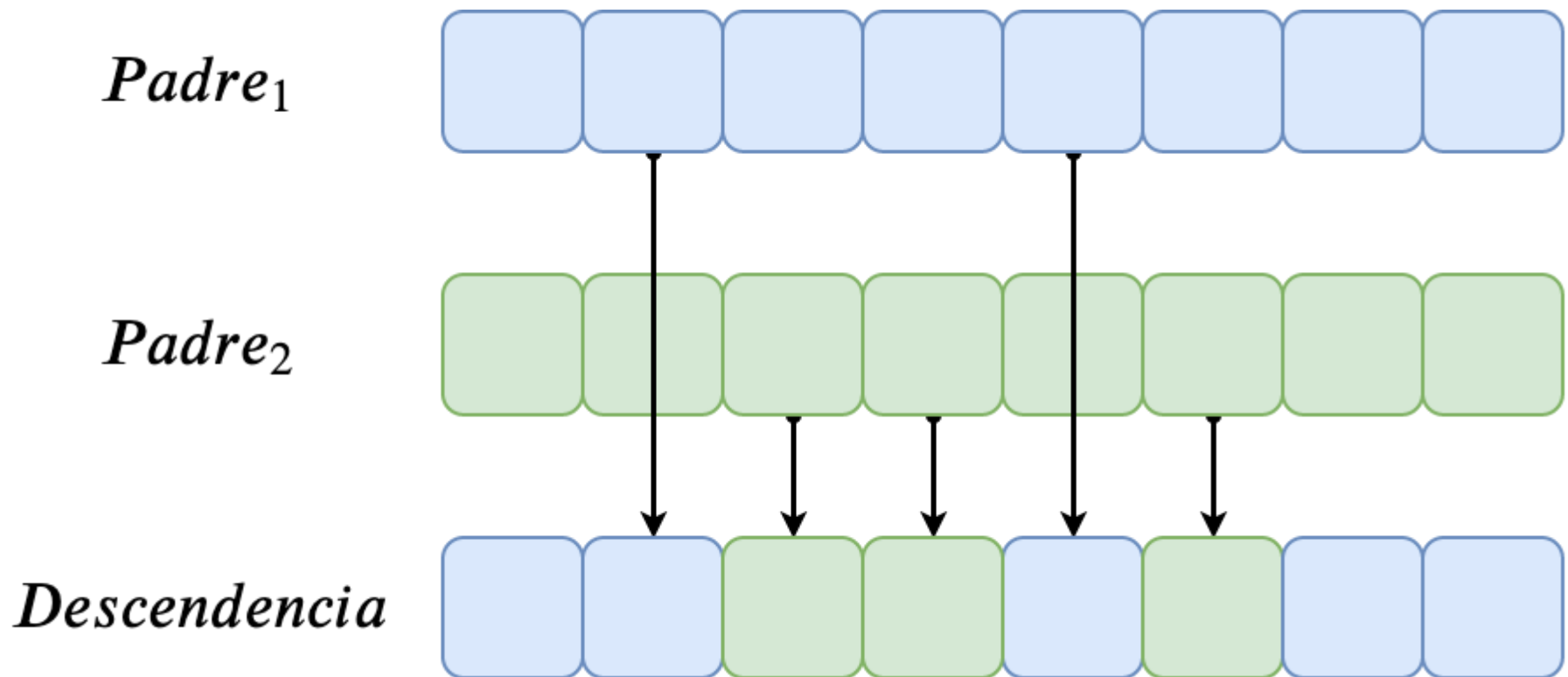
- Ejemplo para  $n = 8$  y  $v = 3$ . Generamos  $r = 6$ . Calculamos el final del segmento como  $((r + v) \bmod n) - 1 = ((6 + 3) \bmod 8) - 1 = 0$ . Tomamos el primer progenitor como  $Padre_1$ :



# AG PAR: Operador de cruce por segmento fijo

---

El resto de genes se seleccionan de manera uniforme de entre los dos padres. Generamos la lista de números aleatorios  $\{1,4\}$ :



# AG PAR: Operador de cruce por segmento fijo

---

**¡El operador de cruce por segmento fijo está sesgado!**

Siempre se seleccionan más genes del padre que se elige como portador del segmento

Si seleccionamos como padre portador del segmento el mejor de los dos estaremos **favoreciendo la explotación**, de forma que la población converge rápidamente

Si seleccionamos como padre portador del segmento el peor padre estaremos **favoreciendo la exploración** y la población tardará más en converger

(Extra: Puede ser interesante hacer un estudio de convergencia de la población variando parámetros del operador de cruce de segmento fijo)

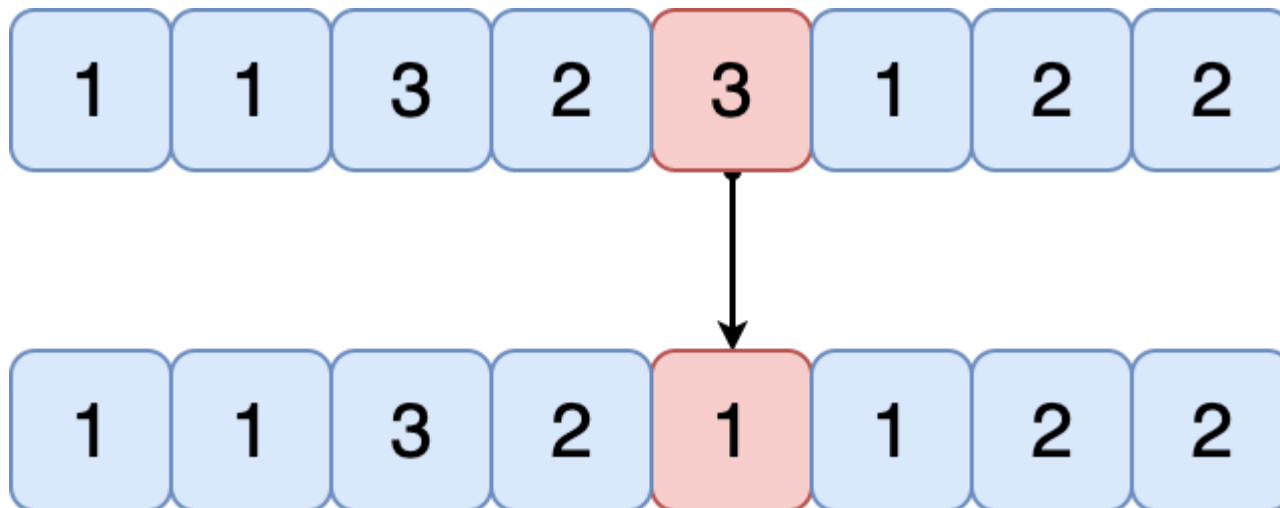


# AG PAR: Operador de mutación uniforme

---

Emplearemos el operador de mutación uniforme

Supongamos que hemos determinado que debe mutar un gen por cada cromosoma en la población. Basta con generar dos números aleatorios, uno para determinar el gen que muta (en el intervalo  $\{0, \dots, n - 1\}$ ) y otro para determinar el nuevo valor (que debe ser distinto del actual **y mantener las restricciones**). Supongamos que muta el gen 4 y que su nuevo valor es 1. El cambio es tan simple como:



# Algoritmo Memético para el PAR: Búsqueda Local Suave

---

Con el objetivo de no desbalancear demasiado el equilibrio exploración-explotación emplearemos una BL suave para incorporarla al esquema de los AGs

Dado un cromosoma, consiste en seleccionar elementos del mismo de forma aleatoria (sin repetición) y asignar a dichos elementos el mejor valor posible con la información disponible

**Recorremos el cromosoma una sola vez**

# AM para el PAR: Búsqueda Local Suave

---

**Algorithm 3:** Búsqueda Local Suave

---

**Input:** Cromosoma (solución)  $S$ , número de clusters  $k$ , número de fallos  $\xi$ .

```
[1]  $RSI \leftarrow \text{RandomShuffle}(\{1, \dots, n\})$ 
[2]  $fallos \leftarrow 0$ ;  $mejora \leftarrow \text{True}$ ;  $i = 0$ 
[3] while ( $mejora$  or  $fallos < \xi$ ) and  $i < n$  do
[4]      $mejora \leftarrow \text{false}$ 
[5]     - Asignar el mejor valor posible a  $S_{RSI[i]}$  (asignar la instancia  $RSI[i]$  al cluster que minimice  $f$ )
[6]     - Si se ha producido cambio en  $S$  significa que ha habido mejora, por lo tanto  $mejora \leftarrow \text{True}$ 
[7]     - Si no se ha producido cambio en  $S$  incrementamos el contador de fallos:  $fallos \leftarrow fallos + 1$ 
[8]      $i \leftarrow i + 1$ 
[9] end
[10] return  $S$ 
```

---

# PAR: Búsqueda Local Suave

---

Cuando la BL no produce cambios en el cromosoma decimos que falla

El contador de fallos está diseñado para evitar que se desperdicien evaluaciones de la función objetivo en cromosomas de mucha calidad. Permitiremos como mucho  $\xi$  fallos en la BL

La condición de parada asegura que si el cromosoma es localmente optimizable, como mucho se recorre una única vez, preservando hasta cierto punto el equilibrio exploración-explotación

Si no lo es, el contador de fallos crecerá rápidamente y la BL se detendrá

# Problemas de Optimización con Algoritmos Meméticos

---

- En los dos problemas (MDP y PAR), emplearemos un AM consistente en un AG generacional que aplica una BL a cierto número de cromosomas cada cierto tiempo
- **En el MDP será necesario pasar de la codificación binaria a la codificación de conjunto de enteros descrita en Seminario 1**
- **Para el PAR emplearemos una nueva BL, no la implementada para la práctica 1**
- Se estudiarán las siguientes tres posibilidades de hibridación:
  - **AM-(10,1.0)**: Cada **10** generaciones, aplicar la BL sobre **todos los cromosomas** de la población
  - **AM-(10,0.1)**: Cada **10** generaciones, aplicar la BL sobre un **subconjunto de cromosomas** de la población seleccionado aleatoriamente con probabilidad  $p_{LS}$  igual a **0.1** para cada cromosoma
  - **AM-(10,0.1mej)**: Cada **10** generaciones, aplicar la BL sobre los **0.1·N mejores** cromosomas de la población actual (N es el tamaño de ésta)

# Problemas de Optimización con Algoritmos Meméticos

---

- Se aplicará **una BL de baja intensidad**. En **MDP se evaluarán sólo 400 vecinos en total en cada aplicación** y en PAR se evaluará la BL explicada tal cual basada en el parámetro  $\xi$ .
- Otras variantes posibles de diseño del AM serían:
  - **AM-(1,1.0)**: En cada generación, aplicar la BL sobre **todos los cromosomas** de la población actual
  - **AM-(1,0.1)**: En cada generación, aplicar la BL sobre un **subconjunto de cromosomas** seleccionado aleatoriamente con  $p_{LS}$  igual a **0.1**
  - **AM-(1,0.1mej)**: En cada generación, aplicar la BL sobre los **0.1·N mejores** cromosomas de la población actual
  - etc.
- Cada una de ellas establece un equilibrio distinto entre exploración y explotación. Se deben hacer experimentos para determinar el ratio óptimo para cada problema

# NOTA SOBRE LOS TIEMPOS DE EJECUCIÓN

---

En los AGs de la segunda práctica no es posible emplear la factorización de la función objetivo empleada en la BL de la primera práctica

Esto se debe a que los operadores de cruce empleados provocan un cambio significativo en las soluciones candidatas generadas y es necesario evaluarlas de forma estándar

**Eso provoca que las ejecuciones de los AGs sean mucho más lentas que las de la BL. Hay que tener este hecho en cuenta y no dejar las ejecuciones para el último momento**