Metaheurísticas

Seminario 3. Problemas de optimización con técnicas basadas en poblaciones

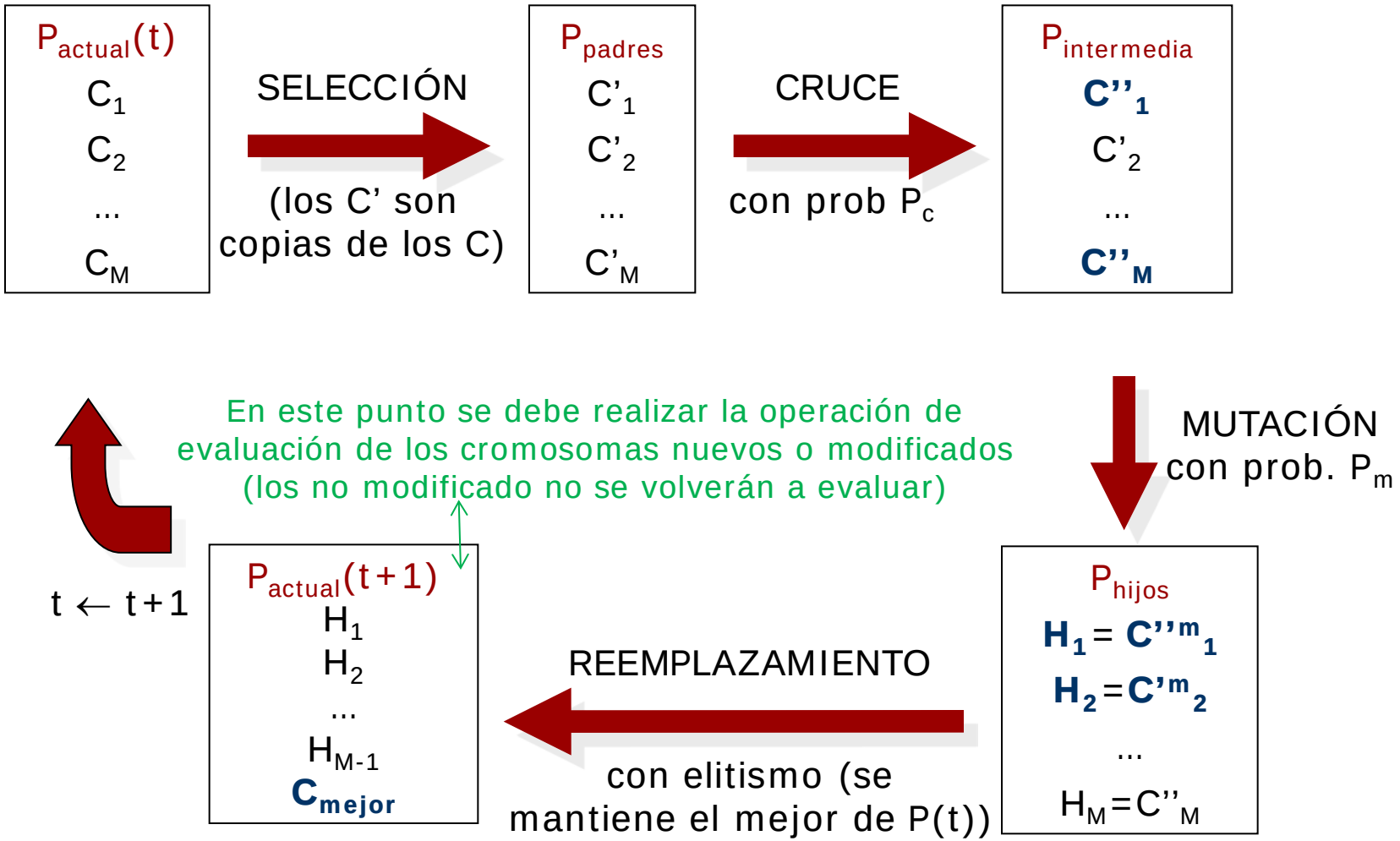
1. Estructura de un Algoritmo Genético y Aspectos de Implementación
2. Problemas de Optimización con Algoritmos Genéticos
   - Asignación de Frecuencias con Mínimas Interferencias (MI-FAP)

Estructura de un Algoritmo Genético

Procedimiento Algoritmo Genético

```
Inicio (1)
    t = 0;
    inicializar P(t);
    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'
        reemplazar P(t) a partir de P(t-1) y P'
        evaluar P(t)
    Final(2)
Final(1)
```

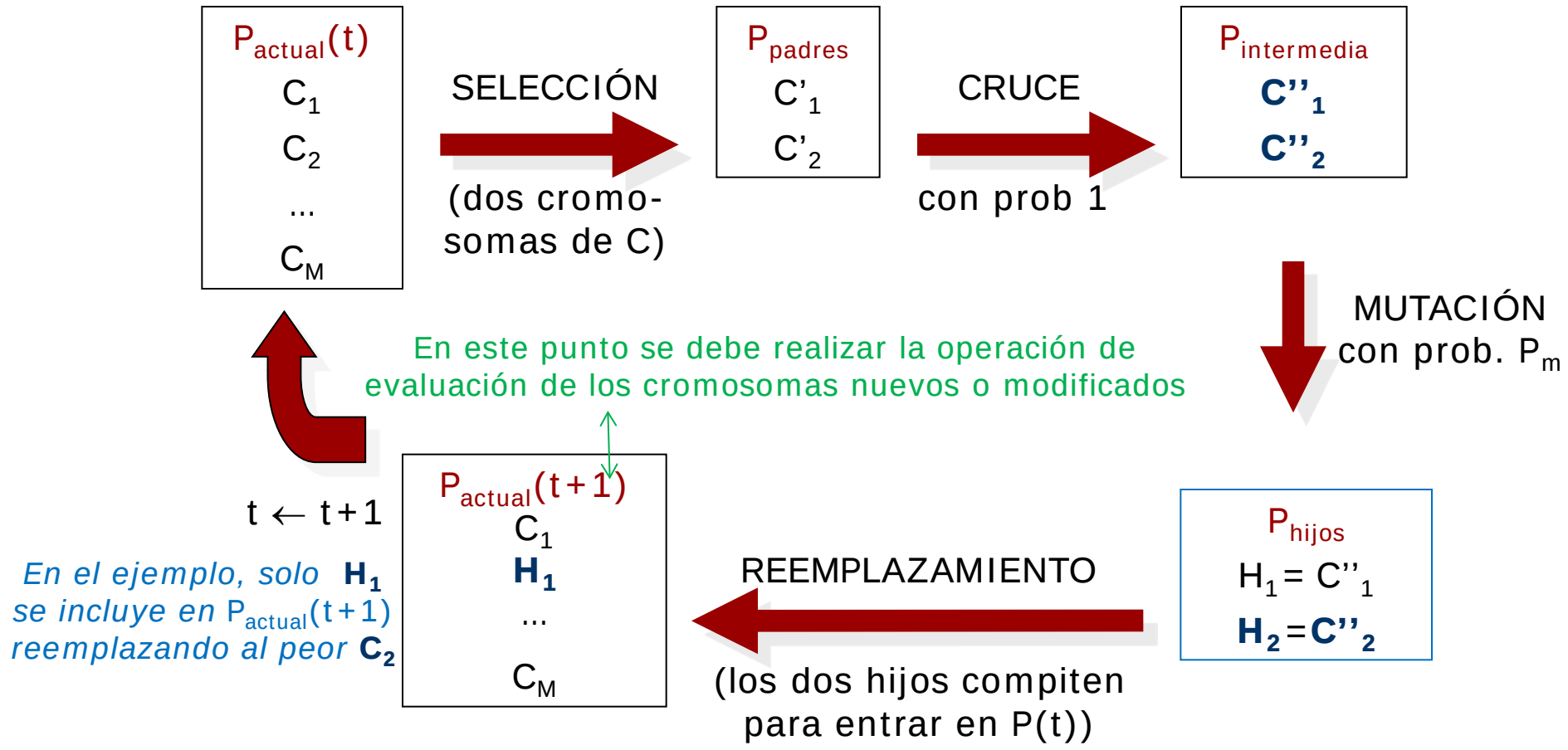
Modelo Generacional:



Elitismo: si la mejor solución de la generación anterior no sobrevive, sustituye directamente la peor solución de la nueva población

Los hijos generados en el cruce reemplazan a los padres, aunque no sean mejor que ellos

Modelo Estacionario:



Los dos descendientes (H_1, H_2) generados tras el cruce y la mutación sustituyen a los dos peores de la población actual $P_{actual}(t)$, en caso de ser mejores que ellos. La nueva población $P_{actual}(t+1)$, está compuesta por los individuos de la población anterior y los nuevos hijos que sustituyen a los peores (en caso de ser mejores que los peores).

Modelo Generacional: Aspectos de Implementación

- Lo más 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 implementaciones eficientes 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 componente aleatoria, 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 50 cromosomas (25 parejas) y una P_c de 0.7, cruzarán 0,7*25= 17,5 (18 parejas).

- ✔ De nuevo, consideramos la aleatoriedad que ya aplica el mecanismo de selección y cruzamos siempre las *Nº 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 más 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·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 · *nº 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 *Nº esperado mutaciones* en cada generación
- ✔ *En el modelo generacional el nº esperado mutaciones es 1 (50*0,02)*

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, …, M} para escoger el cromosoma y otro en {1, …, n} para el gen

- ✔ Existen también mecanismos más avanzados que permiten escoger el gen que muta generando un único número real en [0,1] y haciendo unas operaciones matemáticas

- ✔ En nuestro caso, se obtiene un número aleatorio entre el (1 y el nº total de individuos) y ese es el individuo que se muta (probabilidad mutación del cromosoma.

El Problema del MI-FAP

- El **problema de la Asignación de Frecuencias con Mínimas Interferencias (MI-FAP)** consiste en asignar frecuencias a una serie de trasmisores (los cuales poseen un rango de frecuencias disponibles en las que transmitir). El objetivo es minimizar las interferencias entre las frecuencias.
- Problema de la Asignación de Frecuencias (FAP) es un problema clásico. Existen diferentes variantes del problema dependiendo del objetivo:
  - Minimum Order FAP (MO-FAP): minimizar las frecuencias que se usan en la red
  - Minimum Span FAP (MS-FAP): minimizar la diferencia entre la frecuencia más alta y más baja
  - Minimum Blocking FAP (MB-FAP): minimizar el bloqueo en la red
  - Minimum Interference FAP (MI-FAP).

El Problema del MI-FAP

Definición del problema

- Sea T = $\{t_1, t_2, \ldots t_n\}$ un conjunto de n trasmisores (TRX) y sea $F_i = \{f_{i1}, f_{i2}, \ldots f_{ik}\}\} \subset N$ un conjunto de frecuencias válidas que pueden ser asignadas a los transmisores $t_i \in T$, i=1.. , n. Nótese que el cardinal de k de F_i no es necesariamente el mismo para todos los transmisores. Se define una matriz de interferencias entre las frecuencias MI.

- La función objetivo se puede definir de la siguiente forma:

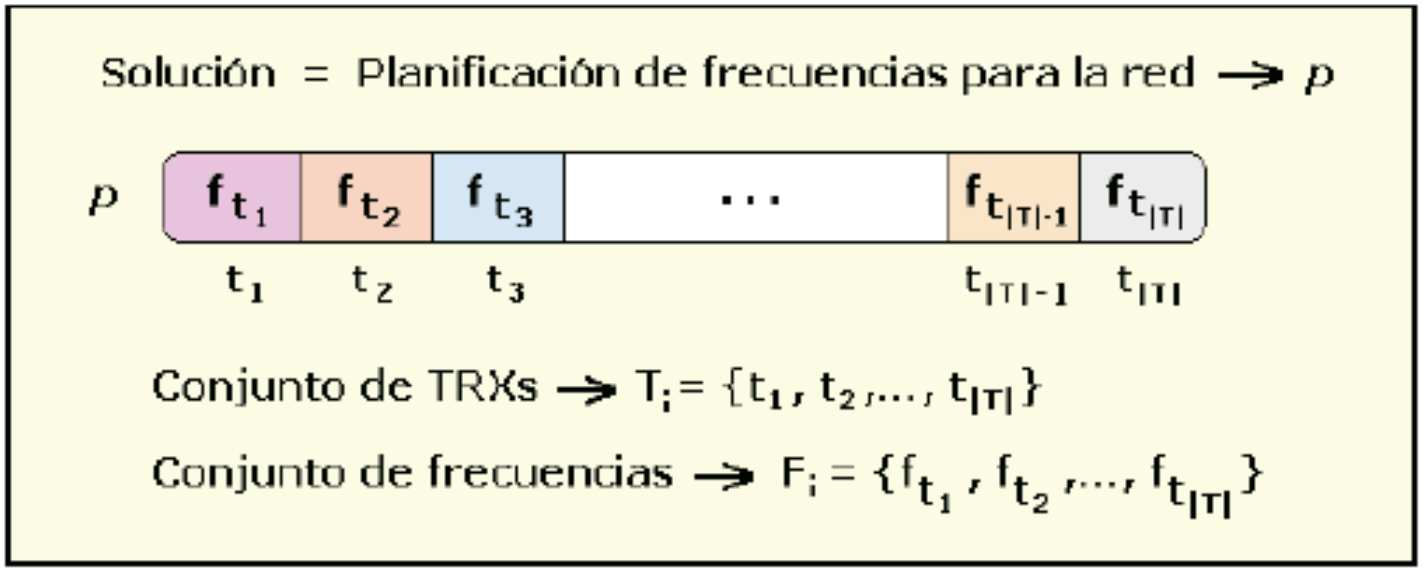
$$fitness = \sum_{i=0}^{n} \sum_{j=i+1}^{n} MI(TRXi, TRXj)$$

- Restricciones: Cada trasmisor solo puede usar una frecuencias de las disponibles para ese trasmisor.

El Problema del MI-FAP

Definición del problema

- Representación de una solución (vector X): Entera



Ejemplo*: *con* 5 *TRX

(10,23,45,65,66,88,23)

Algoritmo Genético para MI-FAP

- Codificación individuos: representación entera.

- Generación de la población inicial: Todos los cromosomas se generarán usando un Greedy simple (no se usa el método aleatorio usado la BL de la práctica 1 para generar la solución inicial). Este Greedy es similar al algoritmo que se usa para generar la solución con greedy aleatorizado del Grasp pero sin la parte del aleatorizado, sin usar el sesgo y sin rellenar 10 aleatoriamente (solamente 1).

  - Greedy simple: Consiste en escoger un transmisor al azar, y una frecuencia al azar para ese transmisor. A continuación, completar el resto de la solución con frecuencias que produzcan menos coste en la solución. Es decir, escogiendo el resto de frecuencias teniendo en cuenta que no incremente o incrementen poco el coste de la solución.

Algoritmo Genético para MI-FAP

- Ejemplo con 5 trasmisores

  Individuo 1 -> (**trx1 |trx2 |trx3 | trx4 |trx5**)

  Escoger al azar un transmisor trx3 incluir una frecuencia al azar para ese transmisor(| | 30 | |) completar a partir del trx4 con frecuencias usando greedy simple (no aleatorio). Ej: (,,30,120,) (,,30,120,45) (8,,30,120,45)..

  Individuo 2 -> Escoger al azar un transmisor trx2 incluir una frecuencia al azar para ese transmisor(| 80 | | |) completar a partir del trx3 : (| 80 |15 | |)

  El motivo de no continuar rellenando el greedy por el primer transmisor es generan individuos más diferentes (mayor diversidad).

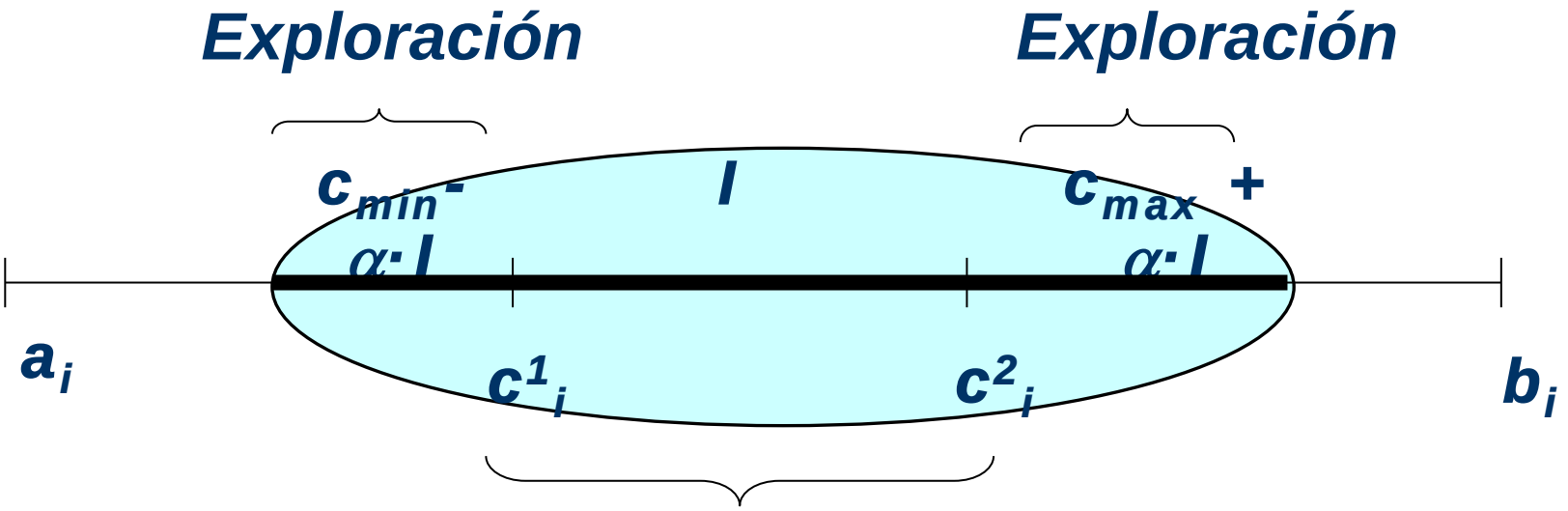
- Modelos de evolución: 2 variantes: generacional con elitismo / estacionario con 2 hijos que compiten con los dos peores de la población

- Mecanismo de selección: torneo binario (Se seleccionan aleatoriamente dos individuos y se comparan sus costes; aquél con menor coste pasa a la siguiente generación). En AGG con cruce BLX se seleccionaran 18 parejas (pág 6)

Algoritmo Genético para MI-FAP

- Tamaño de la población: 50

- Operadores de cruces: El alumno implementara y obtendrá resultados con dos operadores de cruces distintos:
  1. Operador de cruce en 2 puntos (seleccionados aleatoriamente)
  2. Operador de cruce BLX-α.



Algoritmo Genético para MI-FAP

Operador de cruce BLX- α

- Dados 2 cromosomas

  $C_1 = (c_{11},\ldots, c_{1n})$ y $C_2 = (c_{21},\ldots, c_{2n})$,

- BLX- α genera dos descendientes

  $H_k = (h_{k1},\ldots, h_{ki},\ldots, h_{kn})$, $k = 1,2$

- donde h_{ki} se genera aleatoriamente en el intervalo:

  $[C_{min} - I\cdot\alpha, C_{max} + I\cdot\alpha]$

  - $C_{max} = \max \{c_{1i} , c_{2i}\}$
  - $C_{min} = \min \{c_{1i} , c_{2i}\}$
  - $I = C_{max} - C_{min}$,
  - $\alpha \in [0,1]$. En la práctica se utiliza $\alpha = 0,5$

  - **Se escoge la frecuencia más cercana a la obtenida**

Algoritmo Genético para MI-FAP

- Probabilidad de cruce: Será 0.7 en el AGG y 1 en el AGE (siempre se cruzan los dos padres).

- Operador de mutación: consiste en seleccionar al azar un individuo (teniendo en cuenta la probabilidad de mutación), y cambiar ***k*** genes: cambiando la frecuencia de ese gen (TRX) por otra de las disponibles para ese TRX. Donde *k* esta determinado por la probabilidad de mutación por gen (en concreto 0,1)
- Probabilidad de mutación por cromosoma: *será de 0.02. En el modelo generacional el nº esperado mutaciones es 1 (50*0,02). Es decir se selecciona un individuo y ese es el que se muta.*

- Reinicialización: Se aplicara una reinicialización para evitar un estancamiento de la búsqueda. Será dinámico y se podrán hacer 0, 1 o muchas veces, dependerá de la evolución del algoritmo.

Algoritmo Genético para MI-FAP

- En concreto se aplicara una reinicialización cuando se cumpla una de las siguientes condiciones:
  - Después de 20 generaciones no se haya conseguido mejorar la mejor solución encontrada hasta el momento, para el AGG. No se considera en el AGE.
  - O en el caso de que la población converja hacia una misma solución, es decir, que el 80% de la población contengan el mismo individuo. Para ambos algoritmos AGG y AGE.

- La forma de realizar la reinicialización es incluir la mejor solución y el resto de soluciones se generaran con greedy siguiendo el mismo proceso usado para generar la población inicial.

- Número de evaluaciones: 20000