Metaheurísticas

Seminario 3. Problemas de optimización con técnicas basadas en poblaciones

1. Estructura de un Algoritmo Genético/Memético y Aspectos de Implementación

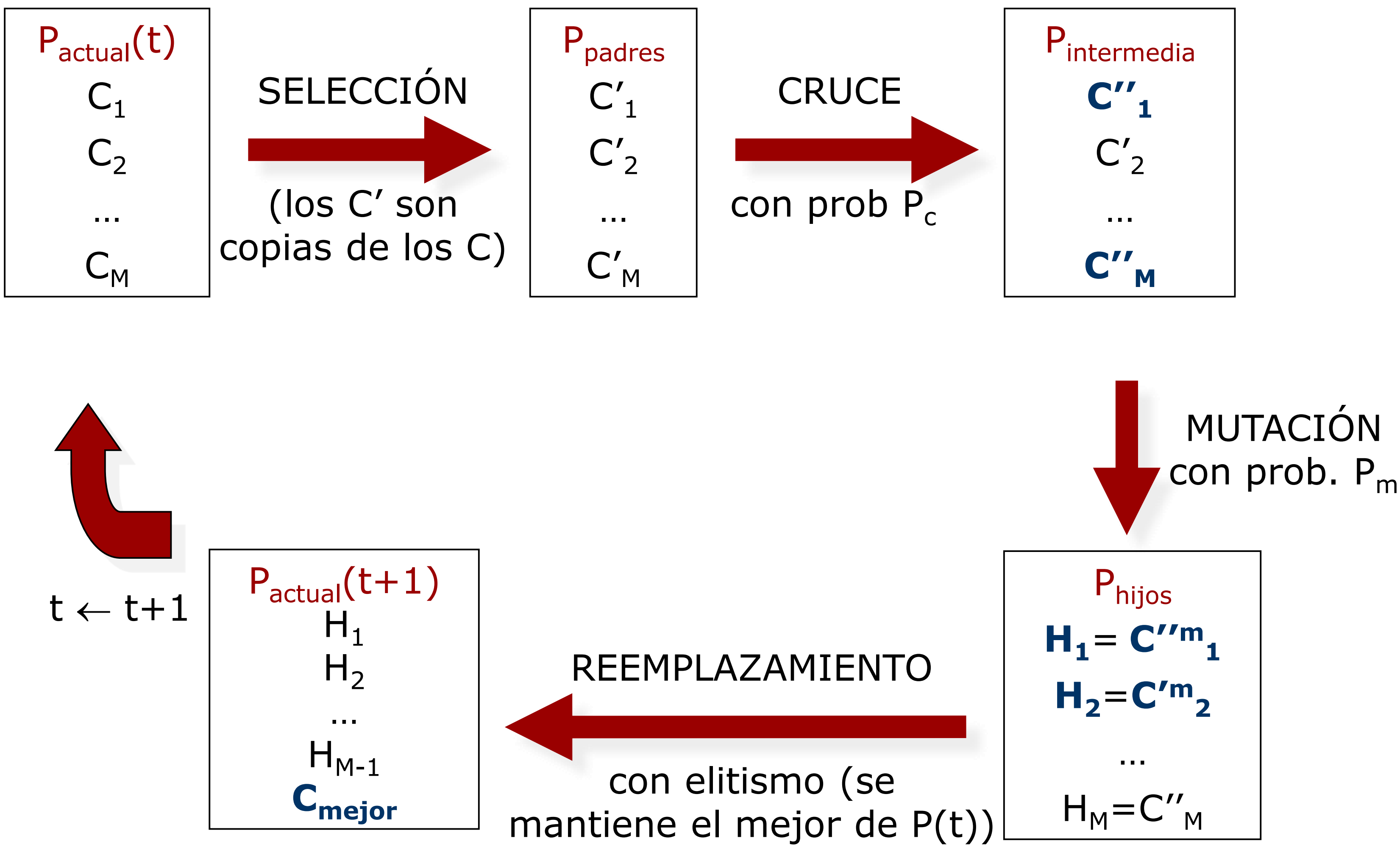
2. Problemas de Optimización con Algoritmos Genéticos y Meméticos
   - Máxima Diversidad
   - Aprendizaje de Pesos en Características

Estructura de un Algoritmo Genético

Procedimiento Algoritmo Genético

```
Inicio (1)
    t = 0;
    inicializar P(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



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 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

- <u>Decidir si una pareja de padres cruza</u>: 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):

$$\textit{Nº esperado cruces} = P_c \cdot M/2$$

- Por ejemplo, con una población de 60 cromosomas (30 parejas) y una P_c de 0.6, cruzarán 0,6*30= 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 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·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

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 a mutar generando un único número real en [0,1] y haciendo unas operaciones matemáticas (ver código entregado en prácticas)

Aspectos de Diseño de los Algoritmos Meméticos

- 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 generación del AG o
     - cada cierto número de generaciones

     y sobre qué 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 (anchura de la BL) y con qué intensidad (profundidad de la BL)?
   - AMs baja intensidad (alta frecuencia de aplicación de la BL/pocas iteraciones)
   - AMs alta intensidad (baja frecuencia de la BL/muchas iteraciones)

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

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

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

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

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 $Sel=(x_1, \ldots, x_n)$ 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: debe contener exactamente *m* 1's

- 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

- Operador de mutación: Intercambio. Se intercambia el valor del gen a mutar x_i por el de otro gen x_j escogido aleatoriamente con el valor contrario

Algoritmo Genético para el MDP

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 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_1 = (0 1 1 0 0 0 1 1 1)
Padre_2 = (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

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
  calculate a violation $v := m - |S_1|$;
  if $v = 0$ **then return** x;
  else if $v < 0$ **then**
    repeat
      find j with $g_j = \max_{j \in S_1} g_j$;
      $x_j := 1 - x_j$, $S_1 := S_1 \backslash \{j\}$, and update gains g;
    until $\sum_{i=1}^{n} x_i = m$;
    return x;
  else
    repeat
      find j with $g_j = \max_{j \in S_0} g_j$;
      $x_j := 1 - x_j$, $S_0 := S_0 \backslash \{j\}$, and update gains g;
    until $\sum_{i=1}^{n} x_i = m$;
    return x;
  endif
end;

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

Si sobran elementos, se van eliminando los de mayor contribución hasta que *x* sea factible

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

Algoritmo Genético para el MDP

Operador de cruce 2: Cruce basado en posición

- 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$_1$ = (0 1 1 0 0 0 1 1 1)
Padre$_2$ = (1 0 1 1 1 0 1 0 0)
Hijo' = (* * 1 * * 0 1 * *)

Restos Padre$_1$: {0, 1, 0, 0, 1, 1} $\rightarrow$ Orden aleatorio$_1$: {1, 1, 0, 0, 1, 0}
Orden aleatorio$_2$: {0, 1, 0, 1, 0, 1}

Hijo$_1$ = (1 1 1 0 0 0 1 1 0)
Hijo$_2$ = (0 1 1 0 1 0 1 0 1)

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

Algoritmo Genético para el Aprendizaje de Pesos en Características

- Representación real: un vector real $W=(w_1, \ldots, w_n)$ en el que cada posición *i* representa el peso que pondera la característica *i*-ésima y su valor en [0, 1] indica la magnitud de dicho peso

- Generación de la población inicial: aleatoria con distribución uniforme en [0, 1]

- 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: Cruce BLX-0.3 y cruce aritmético

- Operador de mutación: El operador $Mov(W,\sigma)$ de Mutación Normal (diapositiva 64 del Seminario 2)

Algoritmo Genético para el APC

Cruce BLX-α con α=0.3

- Dados 2 cromosomas

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

- BLX- α genera dos descendientes

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

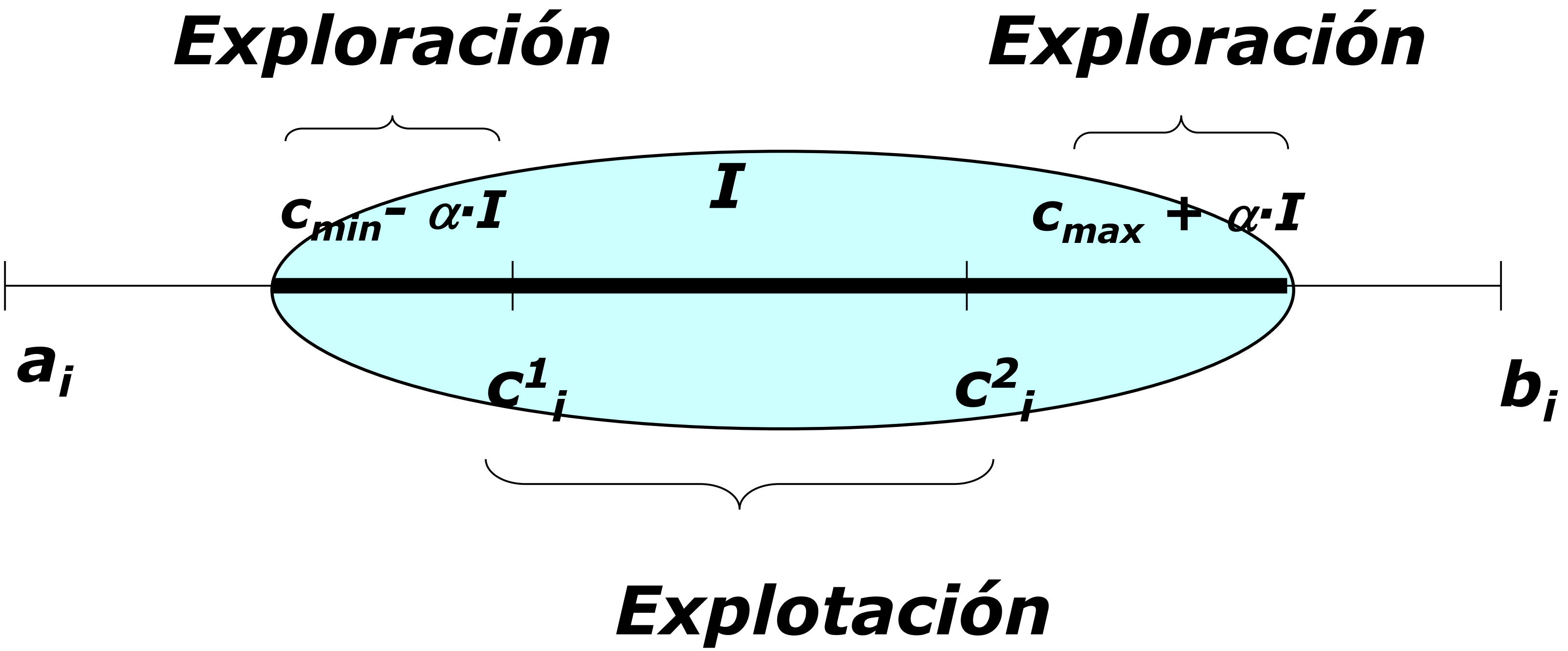
- donde h_{ki} se genera aleatoriamente en el intervalo:

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

  - $C_{max} = \max \{c_{1i} , c_{2i}\}$
  - $C_{min} = \min \{c_{1i} , c_{2i}\}$
  - $I = C_{max} - C_{min}$, $\alpha \in [0,1]$

Algoritmo Genético para el APC

Cruce BLX-α con α=0.3



Algoritmo Genético para el APC

Cruce basado en la media aritmética (cruce aritmético)

a	*b*	*c*	*d*	*e*	*f*

A	*B*	*C*	*D*	*E*	*F*

↙

$(a+A)/2$	$(b+B)/2$	$(c+C)/2$	$(d+D)/2$	$(e+E)/2$	$(f+F)/2$

Problemas de Optimización con Algoritmos Meméticos

- En los dos problemas (MDP y APC), emplearemos un AM consistente en un AG generacional que aplica una BL (Seminario 2) 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**

- 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 APC se evaluarán 2·n vecinos en total en cada aplicación, dos por cada componente

- 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