Metaheurísticas

Práctica 3. Metaheurísticas basadas en poblaciones

Pedro Antonio Gutiérrez

Asignatura "Metaheurísticas"
3º Curso Grado en Ingeniería Informática
Especialidad Computación
Escuela Politécnica Superior
(Universidad de Córdoba)
pagutierrez@uco.es

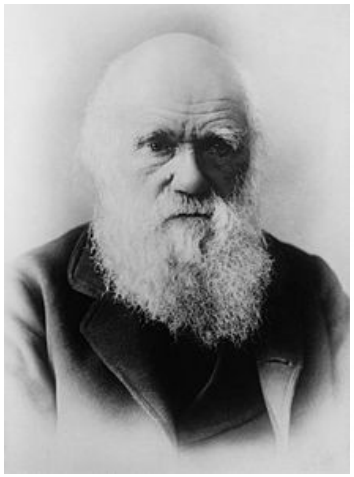
5 de abril de 2014

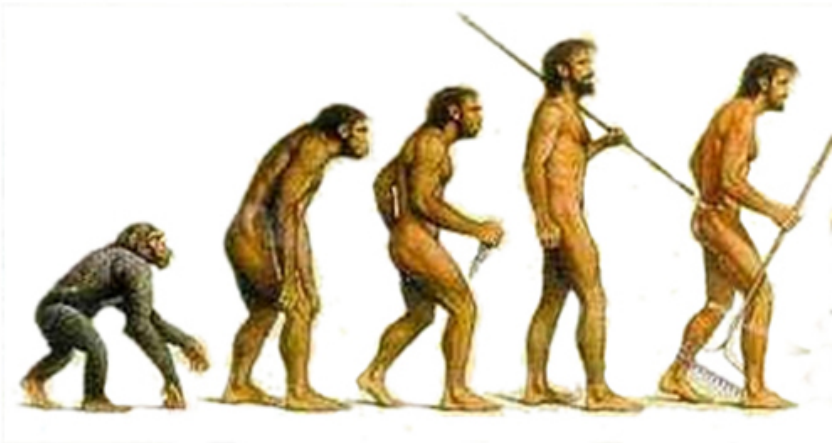
Objetivos de la práctica

- Introducir las metaheurísticas basadas en poblaciones (en concreto los Algoritmos Genéticos, AGs) y aplicarlas al problema del Capacited *p*-Hub (CPH) estudiado durante la primera práctica.
- Familiarizar al alumno con los operadores genéticos típicos de un AG: cruce, mutación, selección y remplazo.
- Estudiar dos modelos específicos de AGs:
  - Algoritmo Genético generacional (AGg).
  - Algoritmo Genético estacionario (AGe).

Algoritmos genéticos

- Se cumplen tres condiciones para que la evolución natural tenga éxito:
  - Un individuo tiene la habilidad de reproducirse.
  - Existe alguna variedad, diferencia, entre los individuos que se reproducen.
  - Algunas diferencias en la habilidad para sobrevivir en el entorno están asociadas con esa variedad.



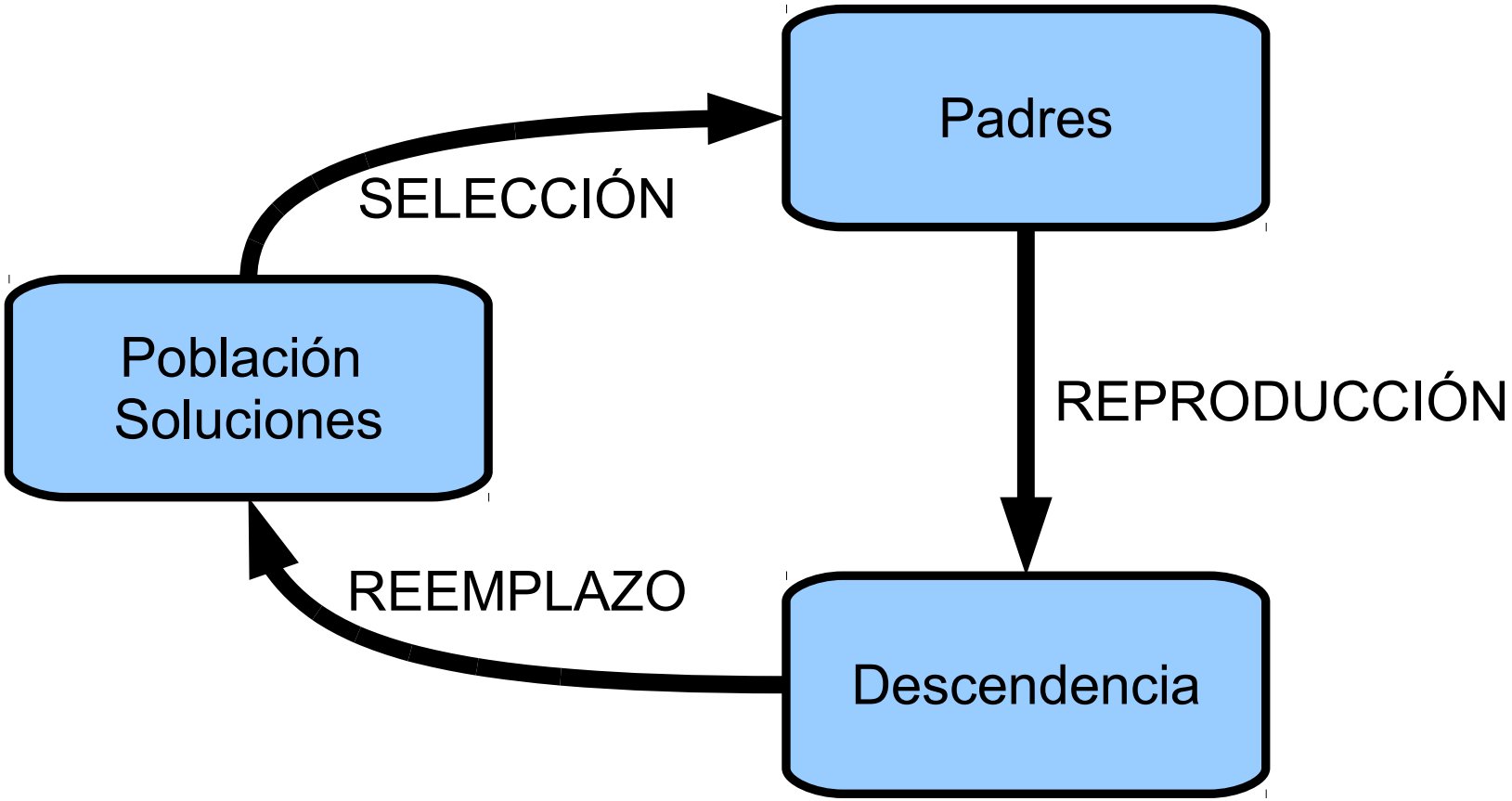


Algoritmos genéticos

- Son algoritmos de búsqueda estocásticos (no deterministas), que incorporan la semántica de la evolución natural a los procesos de optimización.
  - Darwin: Las especies evolucionan acorde al medio y sobreviven los mejor adaptados (1859).
  - Individuos de cada especie $\Rightarrow$ Soluciones al problema.
  - Operadores de mutación y cruce de individuos que modifican las soluciones.

Algoritmos Genéticos	Evolución Natural
Individuo	Solución Candidata
Adaptación	Función Objetivo
Entorno	Problema a resolver

Algoritmo genético básico



Procedimiento algoritmo genético

Inicio

1. $t \leftarrow 0$
2. $P(t) \leftarrow$ generarPoblacionInicial()
3. evaluar($P(t)$)
4. **Repetir**
   1. $P' \leftarrow$ seleccionar($P(t)$) *// Selección y copia de la población*
   2. $t \leftarrow t + 1$
   3. $P' \leftarrow$ cruce(P') *// Cruce con una probabilidad*
   4. $P' \leftarrow$ mutacion(P') *// Mutación con una probabilidad*
   5. evaluar(P') *// Evaluar la descendencia (a los modificados)*
   6. $P(t) \leftarrow$ reemplazo($P(t-1)$,P') *// Construir nueva población*

   Hasta (CondicionParada)
5. **Devolver** mejor solución en $P(t)$.

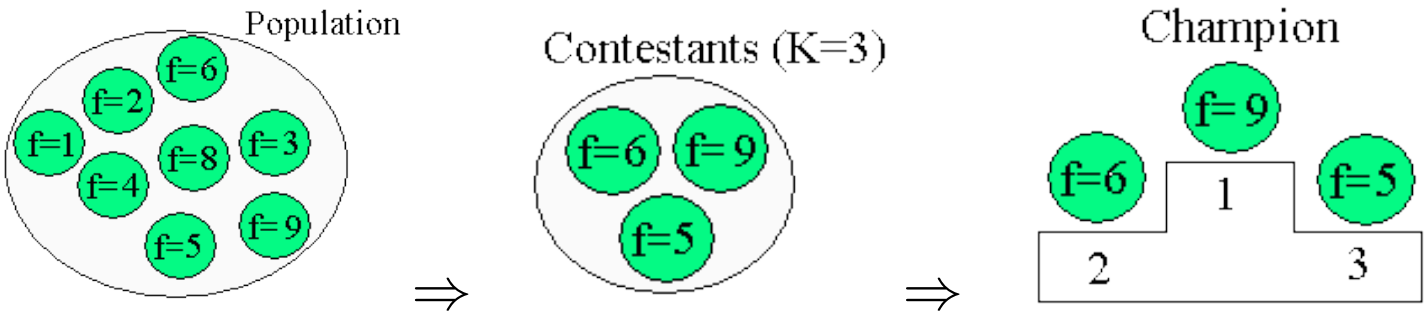
Fin

Algoritmo genético: selección

- Selección:
  - El operador de selección asegura que los mejor adaptados (mejor función objetivo) tienen más posibilidades de reproducirse y generar descendencia.
  - Sin embargo, el operador también debe dejar alguna oportunidad a los peor adaptados, ya que su material genético puede ser bueno en un futuro.
  - Vamos a utilizar dos tipos de selección (los más extendidos):
    - Selección por torneo.
    - Selección por ruleta.
  - Los operadores de selección parten de un conjunto de N individuos y deben seleccionar M representantes.

Algoritmo genético: selección

- Selección por torneo:
  - Parámetro K: tamaño del torneo.
  - Repetir el siguiente proceso M veces:
    - Se escogen aleatoriamente K individuos de entre los N totales.
    - Esos K individuos "concursan", de manera que el ganador es el que tiene mejor aptitud.
    - El individuo ganador pasa a formar parte de los seleccionados.



Algoritmo genético: selección

- Selección por ruleta:
  - Construir una ruleta, de manera que cada individuo tenga una región proporcional a su aptitud (función objetivo).
    - La representamos con un vector $\mathbf{p} = \{p_1, p_2, \ldots, p_N\}$, con tantas posiciones como individuos y con el siguiente valor:

$$p_i = \frac{F_i}{\sum_{j=1}^{N} F_j}, \tag{1}$$

donde F_i es la aptitud acumulada hasta el individuo i-ésimo, es decir:

$$F_i = f_1 + f_2 + \ldots + f_i, \tag{2}$$

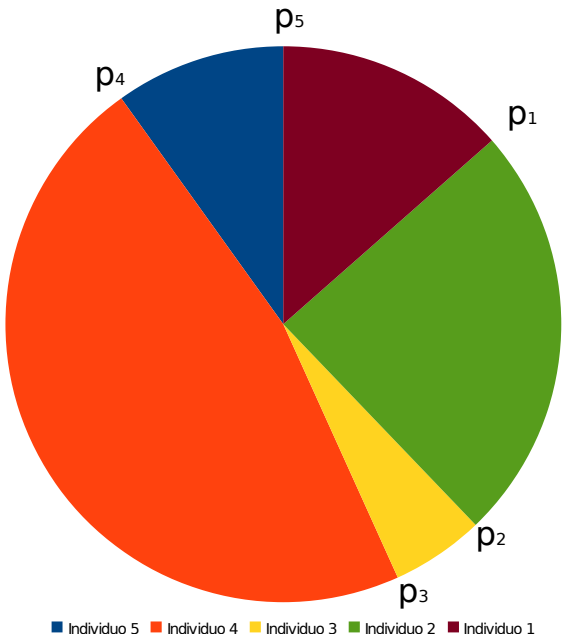
siendo f_i la aptitud de cada individuo.

  - Todo esto suponiendo que la aptitud f_i es mejor cuanto más alta. En caso contrario, podemos tomar $f_i' = \frac{1}{f_i}$.

Algoritmo genético: selección

- Selección por ruleta:
  - Ejemplo de construcción de la ruleta:

i	f_i	F_i	p_i
1	15	15	$15/111 = 0,1351$
2	27	42	$42/111 = 0,3784$
3	6	48	$48/111 = 0,4324$
4	52	100	$100/111 = 0,9009$
5	11	111	$111/111 = 1,0000$



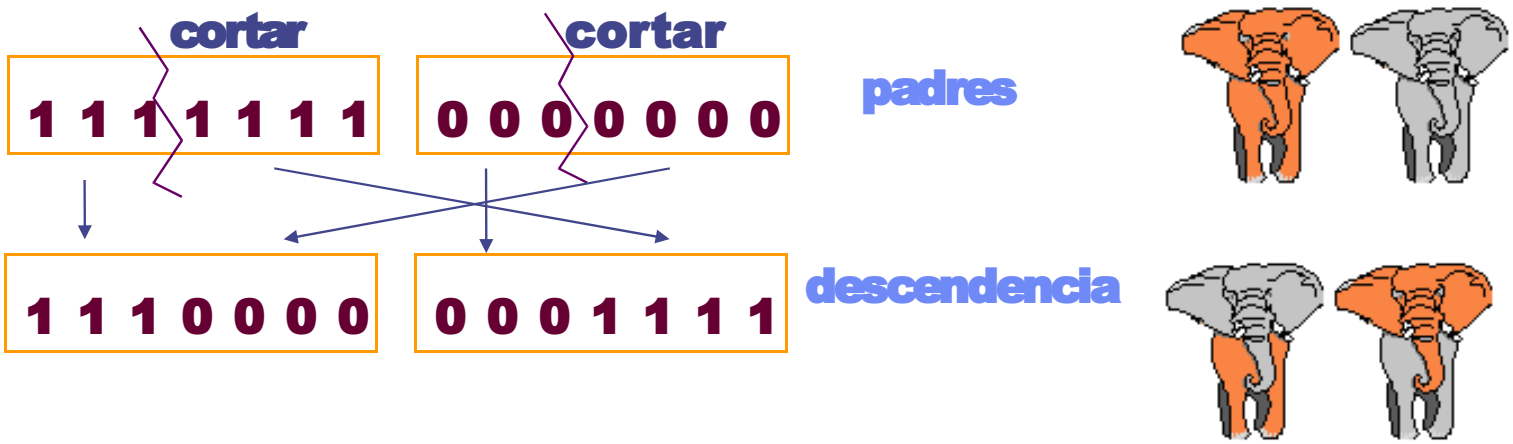
Algoritmo genético: selección

- Selección por ruleta:
  - Una vez construida la ruleta, repetimos los siguientes pasos M veces:
    - Generar un número aleatorio r entre 0 y 1 ($r = U(0, 1)$).
    - Recorrer el vector $\mathbf{p}$ de 1 a N y escoger el primer individuo i tal que:

$$p_i \geq r. \tag{3}$$

Algoritmo genético: cruce

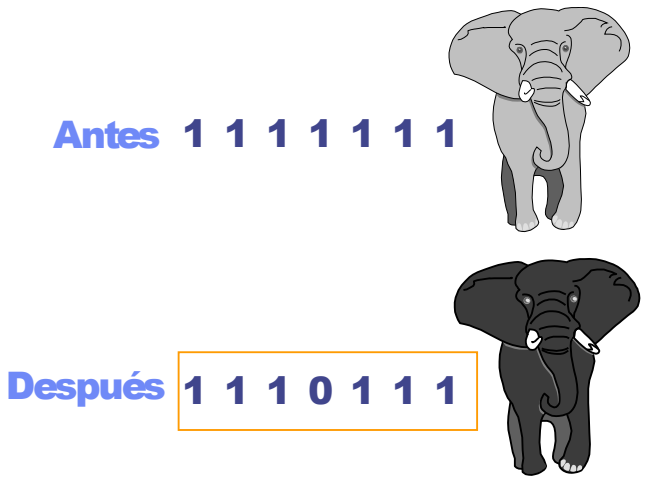
- Operador de cruce:



  - La idea es crear una descendencia que tenga material genético de los dos padres que se cruzan.
  - Por cada individuo de la población de padres (P'), se decide si aplicar o no el cruce con una probabilidad de cruce p_c.
  - El otro padre lo escogemos aleatoriamente de entre los disponibles.
  - Si el operador no se aplica, los hijos no deberían evaluarse.

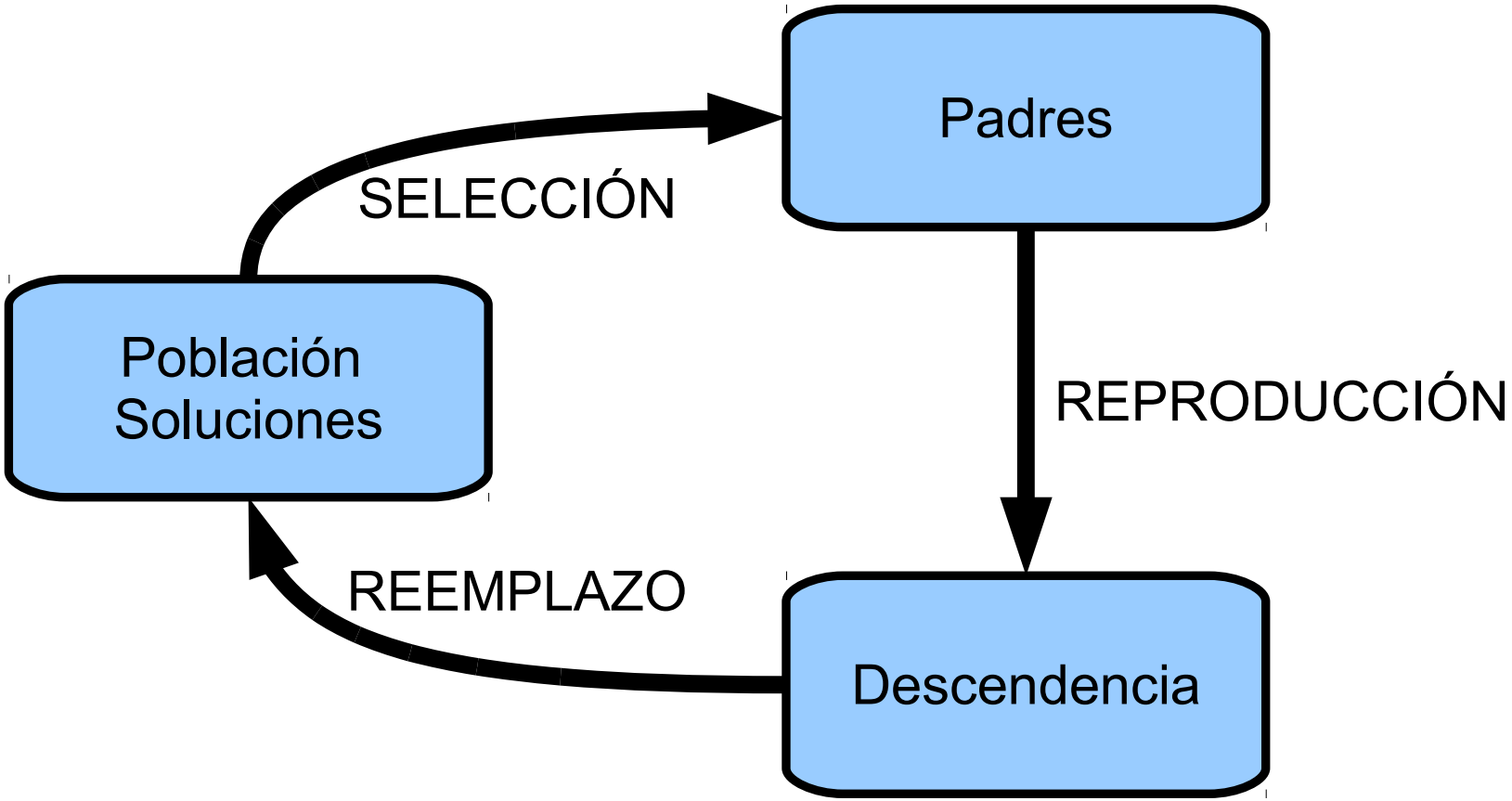
Algoritmo genético: mutación

- Operador de mutación:



  - La idea es modificar sutilmente la solución para introducir nuevo material genético, que puede llevar a mejores resultados.
  - Por cada individuo de la población de padres (P'), se decide si aplicar o no el operador de mutación con una probabilidad de mutación p_m.
  - De nuevo, solo evaluar el individuo si el operador llega a aplicarse.

Algoritmo genético básico



Algoritmo genético: variantes

- Vamos a tratar dos esquemas básicos:
  - Algoritmo Genético generacional (AGg):
    - La población P' (Padres) que se selecciona para aplicar los operadores tendrá $T-1$ individuos (-1 para mantener elitismo).
    - En la fase de reemplazo, P' ya operada (Descendencia) sustituye totalmente a $P(t)$ (manteniendo elitismo).
  - Algoritmo Genético estado estacionario (AGe):
    - P' (Padres) solo contiene dos individuos, los mínimos para aplicar los operadores.
    - Por cada nueva descendencia, la función reemplazo($P(t-1)$,P') une a progenitores y descendientes y sobre el conjunto se aplica una selección adicional para formar la nueva población.

Algoritmo genético: elitismo

- Elitismo:
  - Se mantiene siempre el mejor individuo encontrado en la generación anterior.
  - Mantenemos una variable mejorIndividuoAnterior que actualizamos al evaluar la descendencia P'.
  - Hay que hacer hueco para mejorIndividuoAnterior:
    - Si el algoritmo es AGg, P' tendrá $T-1$ individuos (T: tamaño de la población).
    - Si el algoritmo es AGe, la operación de reemplazo elige $T-1$ individuos en lugar de T.
  - En ambos caso, después del reemplazo, añadimos mejorIndividuoAnterior a $P(t)$.

Algoritmo genético: reinicios de la población

- Reinicio de la población:
  - A veces, cuando han transcurrido muchas generaciones, la mayoría de individuos comparten un genotipo muy parecido.
  - Esto puede reducir la efectividad del algoritmo si se produce muy pronto (convergencia prematura).
  - Una estrategia para evitar esta situación es reiniciar la población, manteniendo al mejor individuo.
  - Para ello, reemplazamos todos los individuos por individuos generados aleatoriamente, e incluimos el mejor individuo que se conocía hasta el momento.
  - Para detectar cuando se produce una convergencia, podemos llevar un contador del número de generaciones sin que haya mejorado el mejor individuo.

Algoritmo genético: aspectos de programación

- Es aconsejable que P' sea una **copia** de los individuos, nunca trabajar con las soluciones originales. Así luego será más fácil aplicar el operador de reemplazo$(P(t-1), P')$ correctamente.
- Es aconsejable que la estructura de la solución tenga un campo con su aptitud.
  - Cuando modificamos la solución con un operador, ponemos la aptitud a una valor imposible (p.ej. -1).
  - Cuando evaluamos la población, solo evaluamos aquellas soluciones que tienen una aptitud igual a -1 y contamos cuantas evaluaciones efectivas se han producido, para detener el algoritmo cuando sea necesario.

Pseudocódigo final AG generacional I

Procedimiento AGg

Inicio

1. $eval \leftarrow nGenSinMejorar \leftarrow 0$
2. $P(t) \leftarrow$ generarPoblacionInicial(T)
3. $eval \leftarrow eval+$ evaluar($P(t)$)
4. **Repetir**
    1. $mejor \leftarrow$ buscaMejor($P(t)$)
    2. $P' \leftarrow$ seleccionar($P(t), T-1$) // *Seleccionar* $T-1$ *individuos*
    3. **Para** cada individuo *ind* en P'
        Si $U(0,1) \leq p_c$
            aplicarCruce(*ind*,individuoAleatorioQueNoSeaInd)
        Fin Si
    Fin Para

Pseudocódigo final AG generacional II

4. **Para** cada individuo *ind* en P'
   Si $U(0,1) \leq p_m$
   aplicarMutacion(*ind*)
   Fin Si
   Fin Para
5. $eval \leftarrow eval+$ evaluar(P')
6. $t \leftarrow t+1$
7. actualizar(*nGenSinMejorar*)
8. **Si** $nGenSinMejorar \geq tope$
   $P' \leftarrow$ generarPoblacionInicial($T-1$)
   $eval \leftarrow eval+$ evaluar(P')
   Fin Si
9. $P(t) \leftarrow P' + mejor$ *// La nueva población es la descendencia más el mejor de la generación anterior*

Hasta ($eval \geq maxEval$)

5. $mejor \leftarrow$ buscaMejor($P(t)$)
6. **Devolver** *mejor*.

Fin

Pseudocódigo final AG estacionario I

Procedimiento AGe

Inicio

1. $eval \leftarrow nSinMejorar \leftarrow 0$
2. $P(t) \leftarrow$ generarPoblacionInicial(T)
3. $eval \leftarrow eval+$ evaluar($P(t)$)
4. **Repetir**
    1. $mejor \leftarrow$ buscaMejor($P(t)$)
    2. $P' \leftarrow$ seleccionar($P(t)$,2) *// Seleccionar 2 individuos*
    3. **Si** $U(0,1) \leq p_c$
       aplicarCruce($P'(1)$,$P'(2)$)
       Fin Si
    4. **Para** cada individuo *ind* en P'
       Si $U(0,1) \leq p_m$
       aplicarMutacion(*ind*)
       Fin Si
       Fin Para
    5. $eval \leftarrow eval+$ evaluar(P')

Pseudocódigo final AG estacionario II

6. $t \leftarrow t + 1$
7. $P' \leftarrow$ seleccionar$(P' + P(t), T - 1)$ *// Juntamos los dos descendientes con toda la población mediante una selección de $T - 1$ individuos*
8. actualizar(*nGenSinMejorar*)
9. **Si** *nGenSinMejorar* $\geq$ *tope*
   $P' \leftarrow$ generarPoblacionInicial$(T - 1)$
   eval $\leftarrow$ *eval*$+$ evaluar(P')
   Fin Si
10. $P(t) \leftarrow P' +$ *mejor*

Hasta (*eval* $\geq$ *maxEval*)

5. *mejor* $\leftarrow$ buscaMejor$(P(t))$
6. **Devolver** *mejor*.

Fin

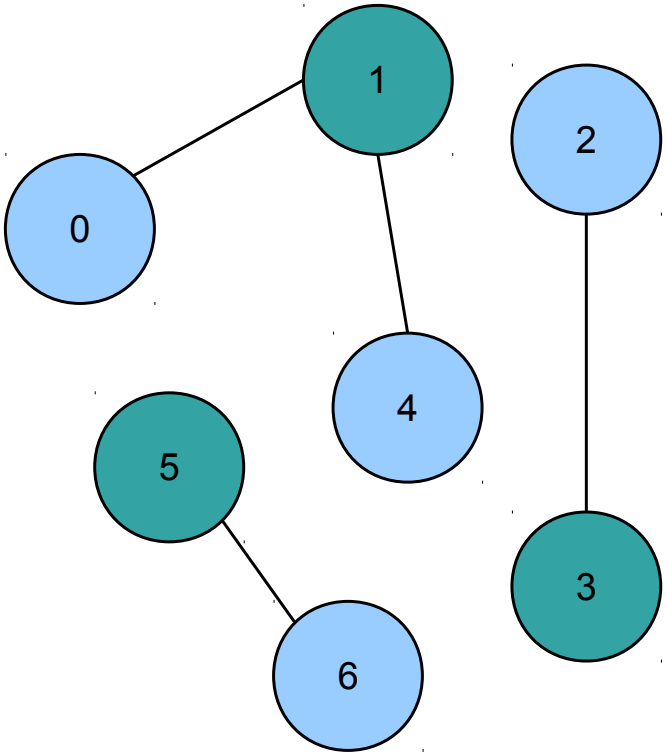
Definición del problema

- El problema se define de la siguiente forma:
  - n centros que pueden ser clientes o concentradores (también denominados *hubs*).
    - p: número máximo de centros que podrían ser concentradores. El resto deben ser clientes.
    - Cada cliente solo puede solicitar servicios de un concentrador.
    - 1 solución: seleccionar los p centros que actuarán como concentradores y decidir el concentrador de cada cliente.
    - Intentando minimizar la suma de las distancias de los clientes y los concentradores.
    - *Capacited*: restricción de que cada concentrador sólo puede servir una determinada cantidad de recursos y que cada cliente tiene una demanda que debe ser satisfecha.
      $\rightarrow$ Solución factible debería asegurarse de que todos los concentradores pueden servir la demanda de recursos de los clientes que tiene.

Representación de la solución

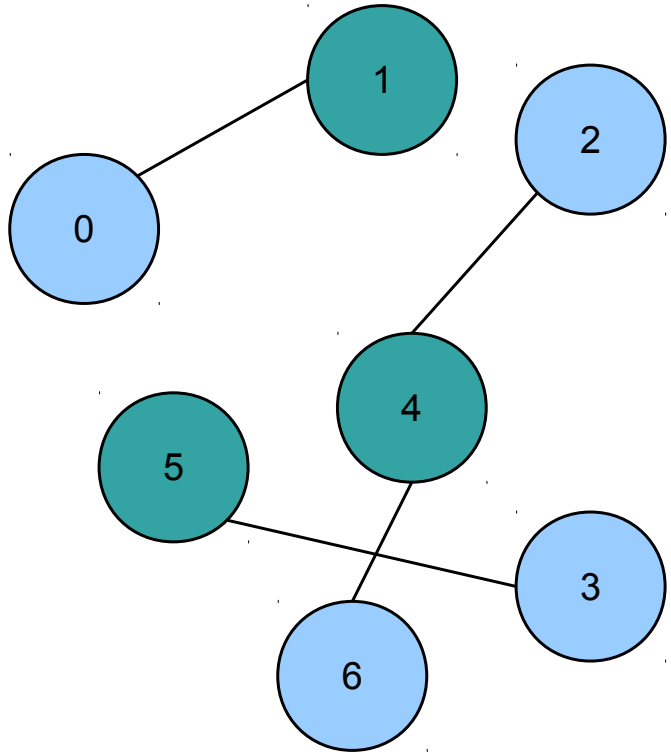
- Vamos a utilizar codificación entera de las soluciones.
- Para facilitar la aplicación del AG, utilizar la siguiente representación:
  - Vector de números enteros **x**, de manera que:
    - Si $x_i < p$: el nodo i-ésimo es un cliente y el valor x_i representa el número de concentrador al que está conectado.
    - Si $x_i \geq p$: el nodo i-ésimo es un concentrador y el valor x_i representa el número de concentrador. En concreto, el número de concentrador sería $x_i - p$ y todo cliente con ese valor estaría conectado a él. Además, el propio concentrador genera una demanda que siempre será atendida por él mismo.
  - Al evaluar la solución, si la restricción de capacidad no se cumple, asignaremos un valor muy malo de aptitud.

Representación de la solución



0	1	2	3	4	5	6
1	4	0	3	1	5	2

4-3=1 3-3=0 5-3=2



0	1	2	3	4	5	6
0	3	1	2	4	5	1

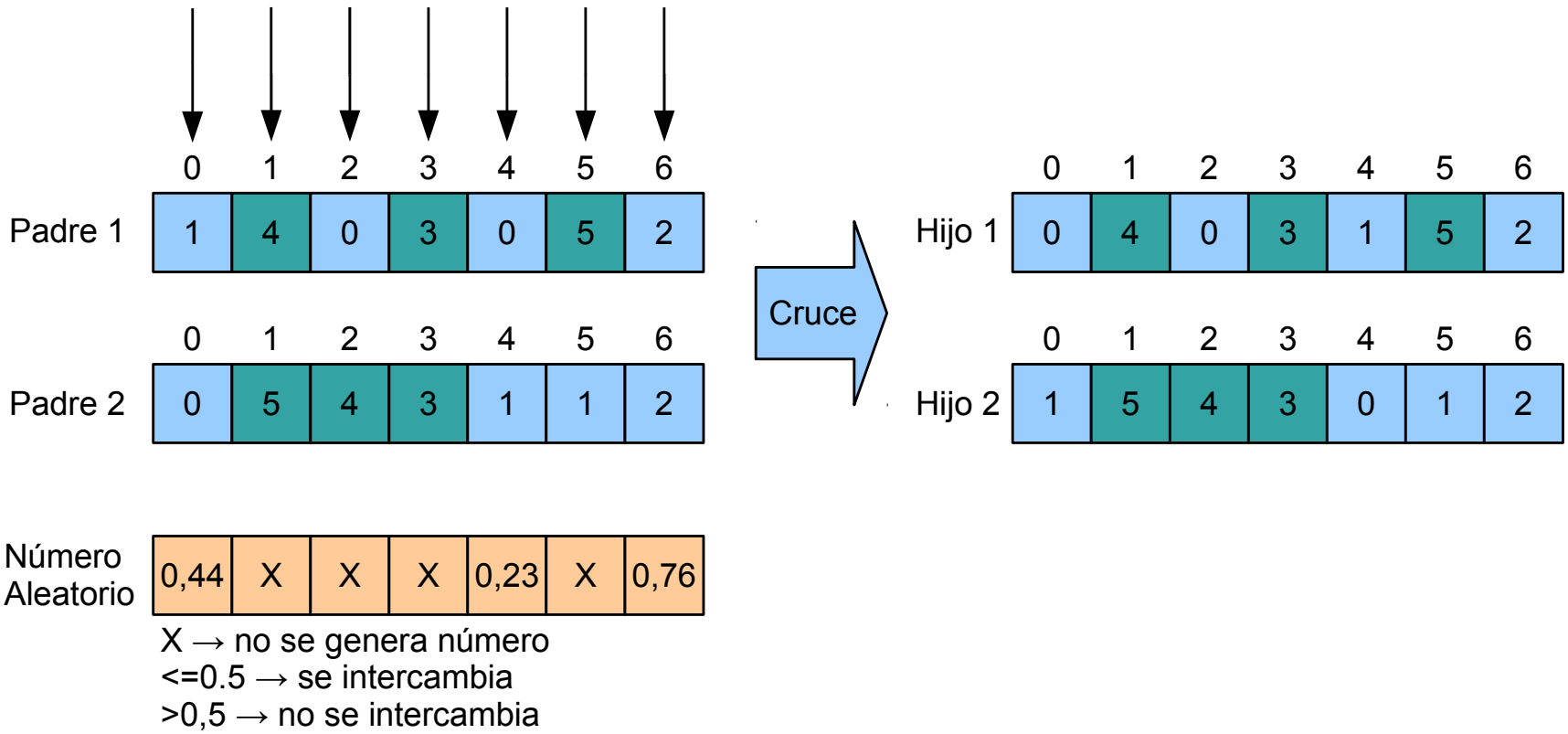
3-3=0 4-3=1 5-3=2

Cruce

- El cruce es peligroso, ya que al intercambiar la información de un gen de dos soluciones, las soluciones generadas pueden ser no válidas.
- Por ello, se aplica un operador muy conservador:
  - Recorremos posición a posición los vectores de los dos padres. Para cada posición i:
    - Si ambos nodos son de tipo cliente ($x_i < p$), con probabilidad 0,5, intercambiamos el contenido de los nodos.

Cruce

Ejemplo ($n = 6$, $p = 3$):

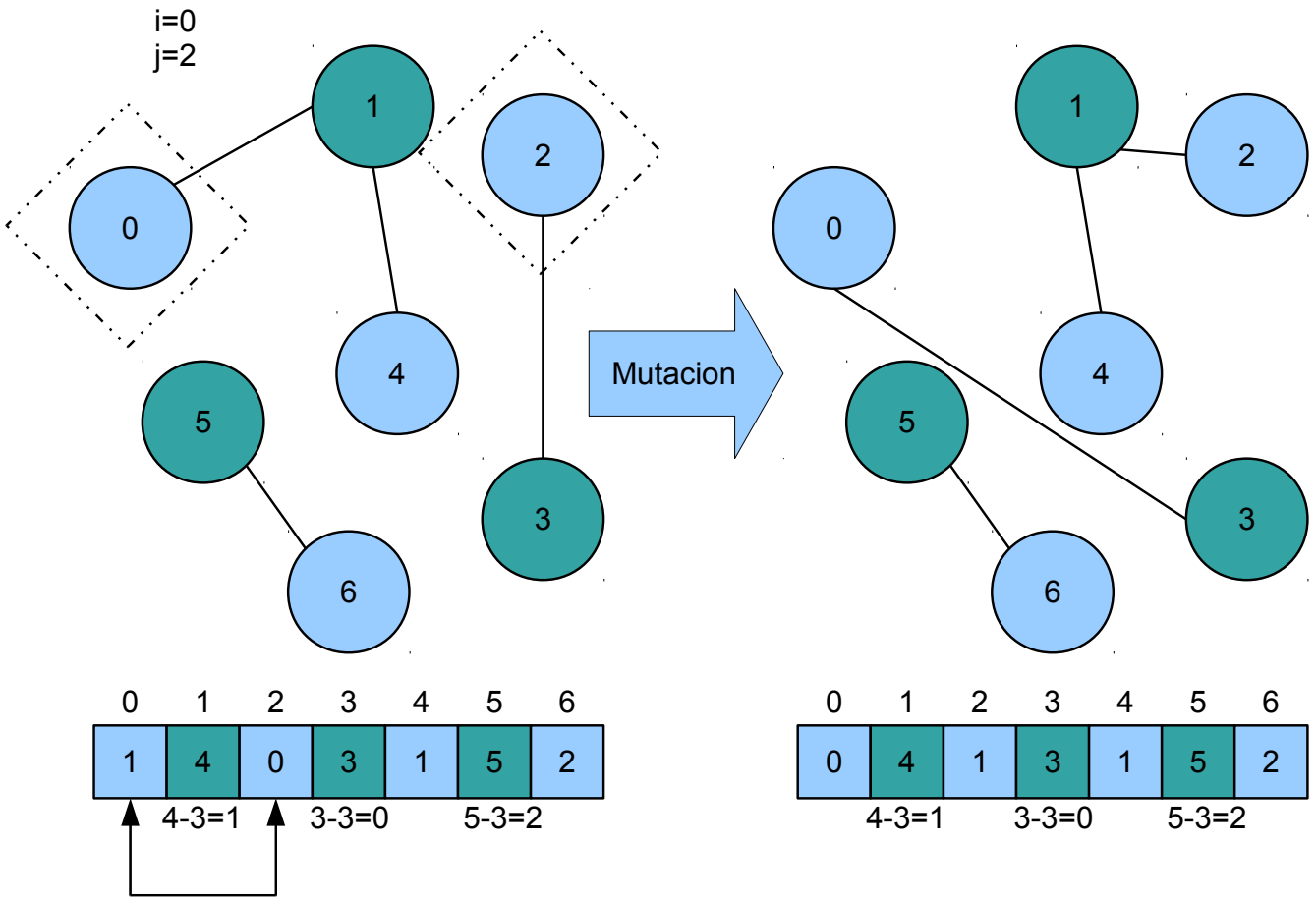


Mutación

- La mutación de una solución se hará de la siguiente forma:
  - Elegimos aleatoriamente dos posiciones del vector (i y j).
  - Intercambiamos el valor de ambas posiciones.
  - Este tipo de mutación respeta siempre las restricciones en cuanto a número de concentradores y en cuanto a que cada cliente esté conectado a un concentrador.

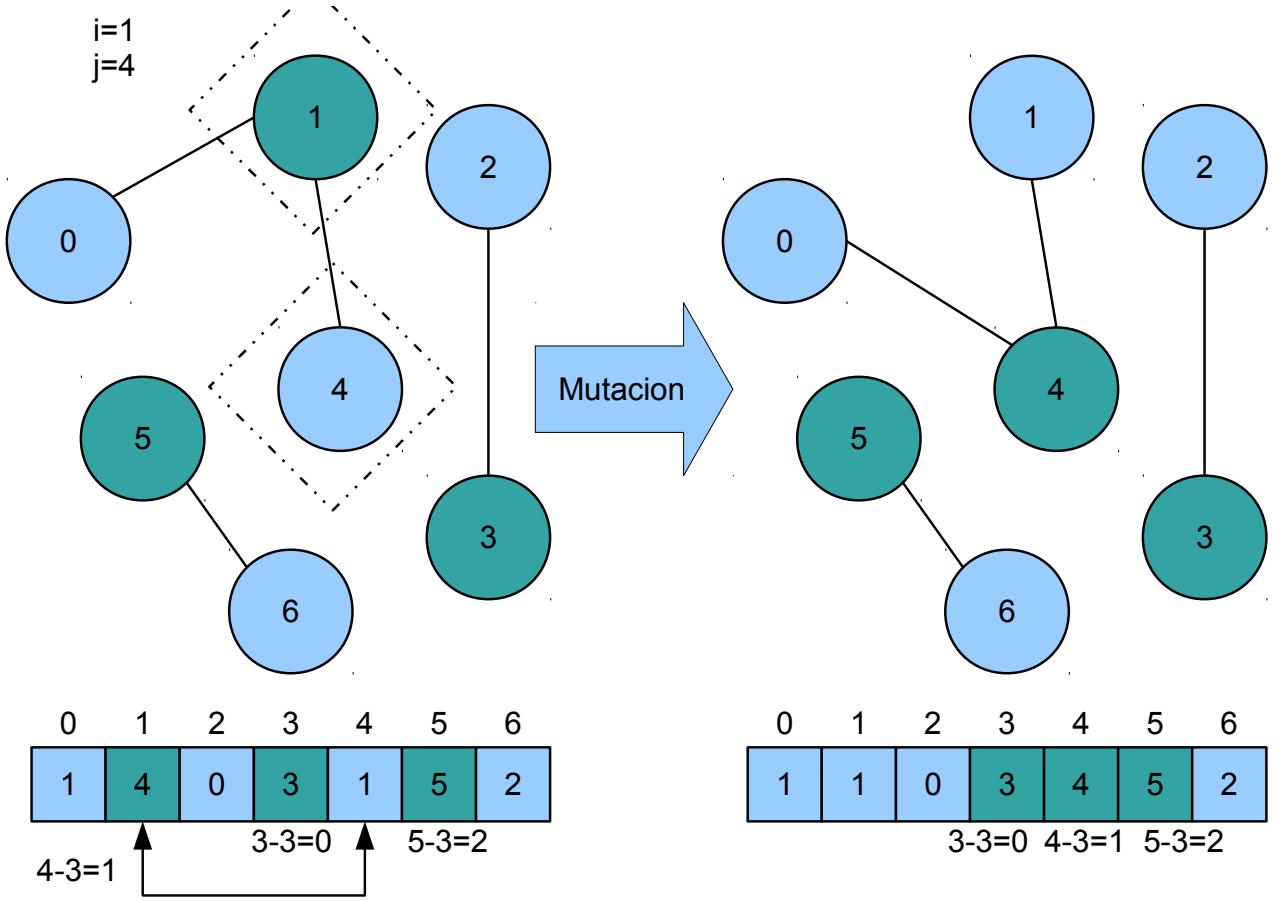
Mutación

Ejemplo ($n = 6$, $p = 3$). Se seleccionan dos clientes:



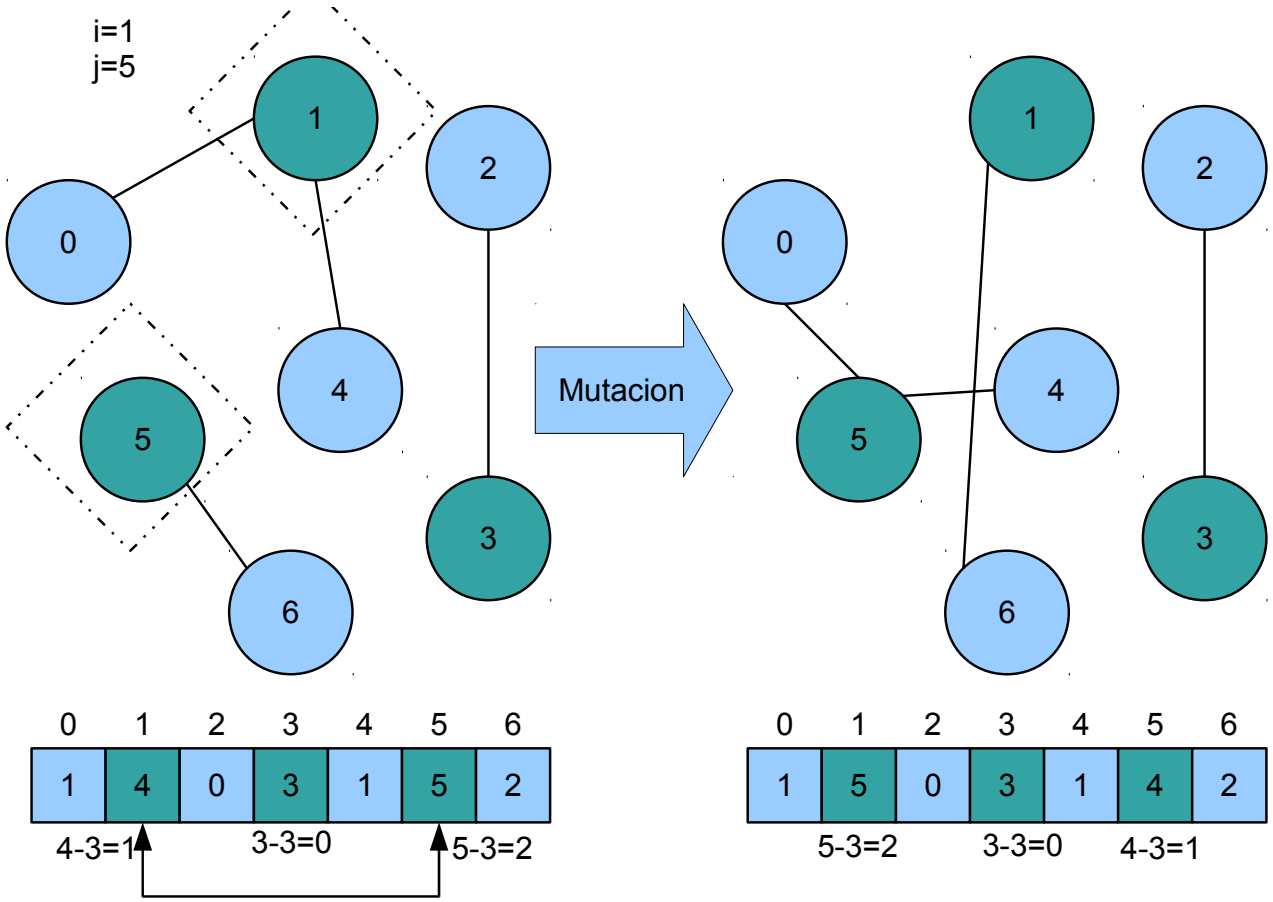
Mutación

Ejemplo ($n = 6$, $p = 3$). Se selecciona un cliente y un hub:



Mutación

Ejemplo ($n = 6$, $p = 3$). Se seleccionan dos concentradores:



Metaheurísticas

Práctica 3. Metaheurísticas basadas en poblaciones

Pedro Antonio Gutiérrez

Asignatura "Metaheurísticas"
3º Curso Grado en Ingeniería Informática
Especialidad Computación
Escuela Politécnica Superior
(Universidad de Córdoba)
pagutierrez@uco.es

5 de abril de 2014