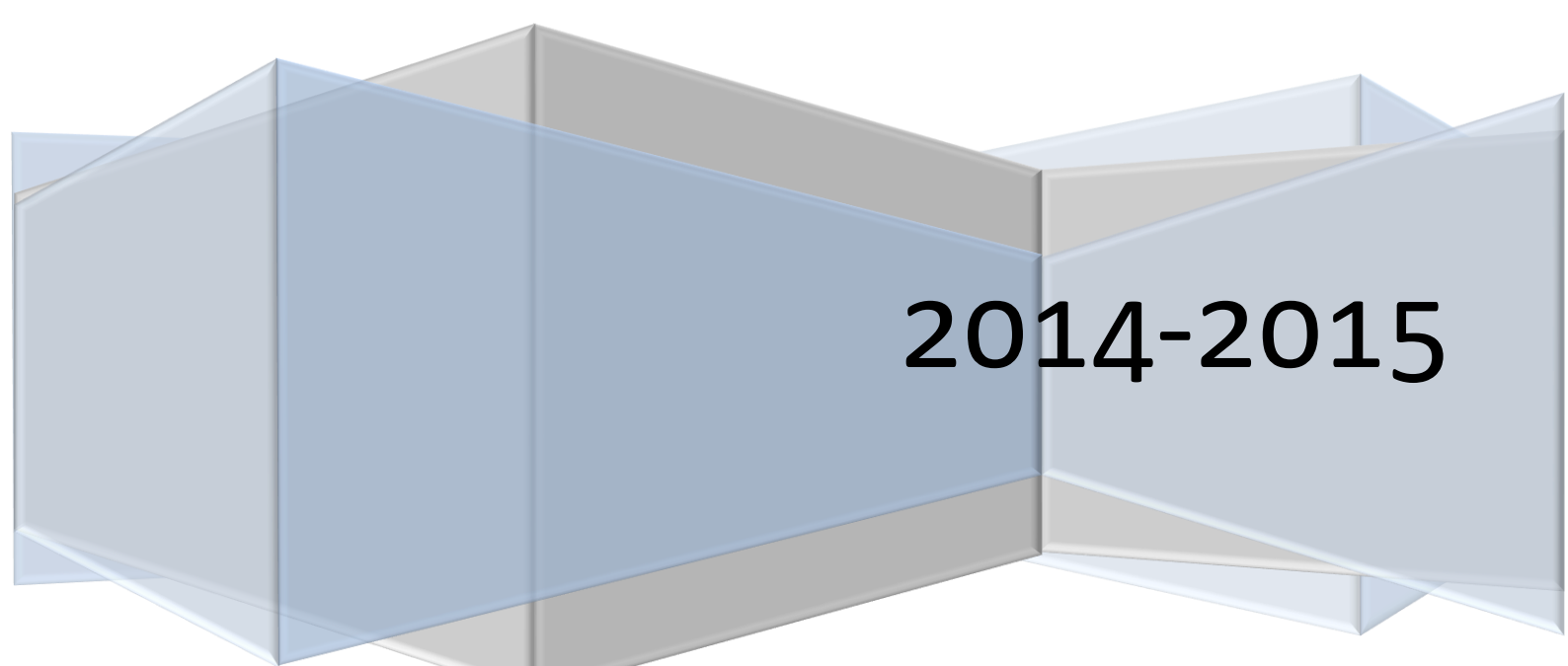


Universidad de Córdoba – Escuela Politécnica Superior  
Grado en Ingeniería Informática

# Práctica 3 - Metaheurísticas

Algoritmos genéticos



2014-2015

# Índice

---

<b>1. Descripción de los problemas.....</b>	<b>1</b>
1.1. CPH	
1.2. Algoritmo Genético	
<b>2. Descripción del esquema de representación y pseudocódigos.....</b>	<b>2</b>
2.1. CPH	
2.2. Algoritmos Genéticos	
<b>3. Descripción de la estructura de los métodos de búsqueda.....</b>	<b>4</b>
3.1. Operador Mutación	
3.2. Operador Cruce	
3.3. Algoritmo Genético estacionario	
3.4. Algoritmo Genético generacional	
3.5. Selección por torneo	
3.6. Selección por ruleta	
<b>4. Experimentos y análisis de resultados.....</b>	<b>6</b>
4.1. Descripción de las instancias	
4.2. Resultados obtenidos	

## 1. Descripción de los problemas

### 4.3. CPH

El problema del capacited p-hub (CPH) consiste en un conjunto de  $n$  centros que podrían tener el rol de cliente o de concentrador (también denominados *hubs*). Hay un número de centros concentradores de forma predeterminada. Se escogen aleatoriamente cuáles van a ser los concentradores. Todos los nodos son clientes, sean o no concentradores, y sólo pueden estar conectados a un concentrador a la vez. Los concentradores tienen una capacidad máxima de oferta, y los clientes una demanda. La suma de las demandas de los clientes de un concentrador no puede superar la oferta máxima que éste tiene.

La solución consistiría en un vector que diría si un hub es concentrador o no. Los hub que no son concentradores están conectados al concentrador cuyo número es el valor que haya almacenado en su posición del vector

### 4.4. Algoritmo Genético

Es un algoritmo de búsqueda estocástico que incorpora la semántica de la evolución natural en los procesos de optimización. Como decía Darwin, las especies evolucionan sobreviviendo los mejor adaptados. En nuestro caso los individuos de la especie son las posibles soluciones del problema. Esta evolución se consigue a través de los operadores de cruce y mutación que modifican los individuos de una generación a otra.

Consiste básicamente en conseguir la mejor solución al problema a través de la generación de posibles soluciones partiendo de una original, a través de la modificación de éstas. Se pueden modificar bien por cruce, es decir, se cogen dos soluciones y de éstas se crean otras dos que se han intercambiado una de las posiciones de dicha solución, o bien a través de la mutación, en la que una de las soluciones simplemente cambia dos posiciones entre sí de su propio vector, siendo el caso del CPH y la representación de la solución que se ha tenido en cuenta.

Este proceso devuelve la mejor de las soluciones que se han producido durante las sucesivas generaciones, por lo que es un algoritmo elitista, siempre se guarda el mejor para que pase a la siguiente generación. Si llega un punto en el que no mejora después de sucesivas generaciones el algoritmo reinicia la población, pero manteniendo la mejor de las soluciones en la población siempre.

## 2. Descripción del esquema de representación y pseudocódigos

### 2.1. CPH

Para este problema se han utilizado tres estructuras, una para almacenar la instancia a utilizar, otra para almacenar la solución y otra para almacenar los datos de un nodo:

- La estructura de la instancia se compone por tres enteros que almacenarán el número de hubs que hay, el número de hubs que son concentradores y la capacidad máxima de éstos. Además hay un vector para almacenar la información de los nodos, una matriz para almacenar las distancias y otro vector para almacenar la demanda que tiene cada uno de los concentradores que soportar.
- La estructura de los nodos se compone de tres enteros: las coordenadas en dos dimensiones del hub y la demanda que éste tiene siendo cliente.
- La estructura de la solución se compone de un vector de enteros que dice si un nodo es concentrador o cliente. Si es cliente señala a qué concentrador está conectado.

#### Función objetivo

Se inicializa el vector de los concentradores a 0.

para(desde inicio (j) hasta P)

    Para(desde inicio (i) hasta N)

        Si(es un elemento del problema) && (no supera la capacidad)

            Se suma su capacidad a la del concentrador

        finSi

    finPara

        cliente++

finPara

Si algún concentrador supera su capacidad ponemos la solución a 100000

Si no supera la capacidad

    Para(desde inicio (i) hasta N)

        Si(x[i] es un cliente)

            Buscamos su concentrador

        finSi

        distancia = distancia + data.distancia[i][indiceConcentrador]

    finPara

finSi

return solucion

Las soluciones que no son válidas se tratan asignando un valor muy alto a la función objetivo, en este caso 100000.

## 2.2. Algoritmos Genéticos

Los individuos de la población inicial se generarán aleatoriamente, de forma que se escogen qué nodos van a ser concentradores de forma aleatoria y posteriormente se van conectando los demás nodos de forma aleatoria sin repetición al nodo que menos carga tiene, siendo el tamaño de la población  $T=200$ .

La condición de parada será que se hayan producido un total de evaluaciones igual a  $5000 \cdot N$ , siendo  $N$  el número de nodos del problema.

La selección de los padres en ambos algoritmos se hará a través de la selección por ruleta, eligiendo  $T-1$  individuos para el AGg y solamente 2 para el AGe. La ruleta se tiene que construir al inverso, al ser un problema de minimización.

El operador de cruce se encarga de cruzar la generación de padres elegida anteriormente. Un individuo tiene una probabilidad de 0.8 de cruzarse con otro individuo aleatorio. Por cada posición en los individuos que son padres habrá una probabilidad de 0.75 de que se intercambie el contenido de dicha posición, es decir, se intercambian el concentrador al que se conectan.

El operador de mutación trabaja de forma similar al anterior, pero sobre un solo individuo. Para que se aplique la mutación hay una probabilidad de 0.2 y consistirá en intercambiar el valor de dos posiciones aleatorias del vector.

Si algún individuo se ha llegado a cruzar o mutar se ha de marcar para su posterior evaluación. La evaluación consiste en volver a calcular la función objetivo para ese nuevo individuo formado.

Si durante 50 generaciones seguidas (AGg) y 500 (AGe) no ha mejorado el mejor individuo de la población, se reinicia la población creando una nueva de forma aleatoria pero de tamaño  $T-1$ . Siempre que se reemplace la población actual hay que mantener el mejor individuo en dicha generación.

## 3. Descripción de la estructura de los métodos de búsqueda

### 3.1. Operador Mutación

elegir nodos de forma aleatoria

intercambiar nodos

marcar para evaluar

### 3.2. Operador Cruce

Para (desde inicio (i) hasta N)

Si (nodo1 y nodo2 (i) son clientes)

Si ( $U(0,1) \leq 75$ )

intercambiar nodos y marcar para evaluar

finSi

finSi

finPara

### 3.3. Algoritmo Genético estacionario

```
Hacer
    Buscar mejor

    Seleccionar 2 por ruleta

    Probar si cruce
    Probar si hay mutacion en los individuos

    Evaluar los que hayan cruzado/mutado (eval++)

    Realizar reemplazo usando seleccion por torneo (manteniendo mejor)

    Comprobar si el mejor ha cambiado

    Si ha cambiado
        nGenSinMejorar=0;
    Sino
        nGenSinMejorar++;
    finSi
    Si (nGenSinMejorar>500)
        Reiniciar poblacion
    finSi

Mientras (eval < evaluacionesMaximas)

Devolver mejor
```

### 3.4. Algoritmo Genético generacional

```
Hacer
    Buscar mejor

    Seleccionar T-1 por ruleta

    Probar si cruce
    Probar si hay mutacion en los individuos

    Evaluar los que hayan cruzado/mutado (eval++)
    Comprobar si el mejor ha cambiado

    Si ha cambiado
        nGenSinMejorar=0;
    Sino
        nGenSinMejorar++;
    finSi

    Si (nGenSinMejorar>50)
        Reiniciar poblacion
    finSi

    Realizar reemplazo sustituyendo poblacion actual y añadiendo el mejor

Mientras (eval < evaluacionesMaximas)

Devolver mejor
```

### **3.5. Selección por torneo**

Coger aleatoriamente individuos para el torneo

Buscar mejor en torneo

Devolver mejor

### **3.6. Selección por ruleta**

Para (desde inicio (i) hasta N)  
total+=valor de element i de poblacion

finPara

total=1/total;

Para (desde inicio (i) hasta N)  
F+= valor de element i de poblacion  
Introducir (F/total) en ruleta

finPara

devolver el primero más pequeño que r

## **4. Experimentos y análisis de resultados**

### **4.1. Descripción de las instancias**

Las instancias de este problema están constituidas de forma que en la primera línea del archivo tenemos en primer lugar el número de nodos del problema, en segundo lugar el número de concentradores que va a hacer y en tercer lugar la carga máxima que van a soportar cada uno de los concentradores.

Seguidamente tenemos los diferentes nodos que componen el problema. Cada línea corresponde a un nodo y está constituida de forma que el primer número es el número del nodo, el segundo su coordenada x, el tercero su coordenada y y el cuarto la demanda que requiere al concentrador que se va a conectar.

Los nombres de las instancias están compuestos por el tipo de problema (phub) seguidos de un número que nos indica el número de hubs del problema y otro que indica el número de concentradores en ese problema. El último número sirve como diferenciador entre los diferentes problemas del mismo tamaño.

### **4.2. Resultados obtenidos**

Los resultados obtenidos se corresponden a los valores de los parámetros que se nos proporcionaban en el enunciado de la práctica, es decir:

- Tamaño de población (T): 200
- Probabilidad de aplicar cruce: 80%
- Probabilidad de aplicar mutación: 20%
- Límite de evaluaciones sin mejora para AGe: 500
- Límite de evaluaciones sin mejora para AGg: 50
- Número máximo de evaluaciones: 5000\*T

	Algoritmo AGg		Algoritmo AGE	
Fichero	Desv	Tiempo	Desv	Tiempo
phub_100_10_10.txt	14,27	7,301	12,12	14,657
phub_100_10_1.txt	20,39	7,752	14,66	19,810
phub_100_10_2.txt	17,90	7,470	18,26	15,033
phub_100_10_3.txt	13,15	7,445	16,26	14,992
phub_100_10_4.txt	13,95	7,694	16,07	15,087
phub_100_10_5.txt	12,52	7,481	15,94	14,774
phub_100_10_6.txt	14,59	7,451	21,39	15,043
phub_100_10_7.txt	17,20	7,160	14,64	14,533
phub_100_10_8.txt	13,49	8,143	18,16	16,548
phub_100_10_9.txt	9,74	7,967	17,38	14,988
phub_50_5_10.txt	9,28	1,436	10,06	3,565
phub_50_5_1.txt	13,62	1,667	18,26	4,026
phub_50_5_2.txt	9,25	1,552	15,68	3,934
phub_50_5_3.txt	18,10	1,651	18,12	4,092
phub_50_5_4.txt	10,74	1,631	14,64	4,058
phub_50_5_5.txt	18,19	1,617	10,23	3,911
phub_50_5_6.txt	9,66	1,721	9,67	4,498
phub_50_5_7.txt	10,11	1,596	10,11	4,071
phub_50_5_8.txt	6,31	1,900	4,43	4,442
phub_50_5_9.txt	7,14	1,565	9,17	4,238
Media	12,98	4,61	14,26	9,81
Desviación Típica	3,96	3,06	4,20	5,99
Máximo	20,39	8,14	21,39	19,81
Mínimo	6,31	1,44	4,43	3,56

**Tabla 1: Resultados**

Como se puede apreciar, con los parámetros dados, se obtienen unos resultados bastante aceptables, acercándose ambos algoritmos a los óptimos globales. Vemos que para instancias más grandes, ambos algoritmos tardan más que con las más pequeñas, lo cual es evidente ya que a mayor número de elementos en una población, más coste computacional se va a tener, ya que se manejan soluciones con muchos más nodos y hay que calcular y recalcular muchas más distancias.

El algoritmo generacional es prácticamente el doble de rápido que el estacionario con mejores resultados en casi todas las instancias, esto se refleja en las medias de las desviaciones de ambos algoritmos con respecto a los óptimos globales. Esta rapidez puede deberse a que el número de evaluaciones máximo sin que haya mejora en la población del generacional es de 50, mientras que en el estacionario es de 10 veces más. El estacionario sólo cambia dos individuos de la población, lo que puede hacer que se quede en un óptimo global y esté mucho tiempo ahí parado hasta que se reinicie la población. En el generacional se intenta aplicar los cambios a casi toda la población lo que hace que haya mucha más diversidad, además de que es más difícil que se quede en un óptimo local ya que se reinicia mucho antes que el estacionario. Estas razones son también por las que se obtienen mejores resultados.



