

Universidad de Jaén

José Morón Rodríguez

DNI: 77349558Y

[jmr00026@red.ujaen.es](mailto:jmr00026@red.ujaen.es)

Grupo de teoría: A

Grupo de prácticas:

8:30 a 10:30



## **Practica 3 Búsquedas Híbridas para el Problema del Clustering**

3º Grado en Ingeniería Informática—Metaheurística

## Índice

.....	0
Esquema de representación.....	3
Calculo de distancias .....	3
Función de evaluación o función fitness .....	3
Generación de la población inicial .....	3
Pseudocódigo .....	3
Operador Cruce.....	4
Operador Mutacion.....	4
Búsqueda Local.....	4
Pseudocódigo .....	4
Algoritmo Genético Generacional.....	5
Pseudocódigo .....	5
Manual de usuario .....	6
Parámetros.....	8
Patrones .....	8
Dimensiones de cada patrón.....	8
Número de cluster.....	8
Semillas .....	8
Resultados obtenidos.....	8
Kmedias.....	8
Análisis de los resultados obtenidos .....	9

## Descripción del problema

Nos encontramos ante un problema de clustering.

Tenemos una serie de datos en unos ficheros, cada fila de ese fichero, es un dato y en cada columna tenemos sus correspondientes componentes.

Esos datos los llamaremos patrones.

Además tenemos la información de en cuantos cluster tenemos que agruparlos.

Con este problema, lo que intentamos conseguir es que los patrones que sean más similares, queden agrupados dentro un cluster distinto a los que sean menos similares, que estarán agrupados en otro cluster.

Para ello nos tenemos que ayudar de algoritmos que no buscan una solución, sino que optimizan una solución que ya tienen, ya sea de forma aleatoria, con un algoritmo greedy...

Estas agrupaciones, las podemos conseguir gracias a que cada cluster, tendrá también sus propias componentes, donde el conjunto de estas componentes, serán los centroides del cluster.

Nos basaremos en el movimiento de un patrón desde un cluster a otro y será mejor el movimiento cuanta menos distancia haya entre ese patrón y el centroide del cluster.

Cada vez que haya un movimiento, los centroides de los cluster deben cambiar, como el calcularlo entero de nuevo puede ser muy costoso, hacemos uso de la factorización.

Para saber si es buena una solución o no, basta con utilizar la función de evaluación, la cual dirá si con los movimientos hechos, hemos mejorado y por lo tanto esa es nuestra mejor solución, o por el contrario, hemos empeorado y es mejor buscar otras soluciones.

# Descripción de la aplicación de los algoritmos

Describimos a continuación explicaremos los distintos elementos del algoritmo AMs.

## Esquema de representación.

Para cargar los patrones en el programa utilizamos un vector de punteros a doubles, los cuales serán los patrones.

Un cromosoma sería la asignación de los M patrones a los K cluster por lo tanto usaremos una representación de orden basada en una permutación que será un vector de tamaño M y que en la posición i un numero de patrón. Siendo los k primeros los centroides (o medoides) y m-k restantes se asignan a cada cluster más cercano. Cada cromosoma se guardara dentro de la población con el resultado de su función fitness.

## Calculo de distancias

Para saber la distancia de un patrón a otro patrón, o de un patrón a un cluster, nos ayudamos de la implementación de la distancia euclídea, que dice lo siguiente:

$$\sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \dots + (x_n - y_n)^2}$$

Siendo  $x_{1...n}$  las componentes del patrón X e  $y_{1...n}$  las componentes del patrón Y.

## Función de evaluación o función fitness

Minimizar la distancia de cada patrón a los cluster.

```

Para cada cromosoma
  Para cada patrón del cromosoma
    Para cada cluster
      Si la distancia es la mínima guardarla
    Fin Para
  Solución=solución+ distancia_minima2
Fin Para
  
```

## Generación de la población inicial

Para cada cromosoma desordenamos un vector con todas las posiciones de los M patrones y calculamos su función fitness:

## Pseudocódigo

```

Para cada cromosoma{
  
```

```

Cromosomai=Desordenamos(cromosoma_modelo)
J=funcionFitnnes(cromosomai);
Insertamos(cromosoma, J);
}

```

## Operador Cruce

Como utilizamos una representación de orden basada en permutaciones, para cada hijo se buscaran los genes que tengan iguales los padres y esos genes no variaran en la nueva generación. El resto de genes se rellenara de forma aleatoria de los restantes. Ejemplo

Padre 1	2	3	6	4	1	8	7	5	10	9
Padre 2	9	2	8	4	3	2	7	10	5	1

Restantes = {1,2,3,5,6,8,9,10}

Hijo 1	6	8	9	4	1	10	7	2	3	5
Hijo 2	3	6	1	4	5	9	7	8	2	10

## Operador Mutacion

Como utilizamos una representación de orden basada en permutaciones, para cada cromosoma se elegirán aleatoriamente dos genes y se intercambiaran. Ejemplo

Cromosoma	2	3	6	4	1	8	7	5	10	9
Cromosoma'	9	2	8	7	3	2	4	10	5	1

## Búsqueda Local

La búsqueda local es un proceso de optimización en el que para cada cromosoma de la poblacion, miramos si ese patrón es más cercano a otro cluster distinto, en cuanto encontramos un cluster más cercano, se considera que se debe cambiar de cluster. Esto se hará mientras que haya cambios o no se hayan hecho 10000 cambios.

## Pseudocódigo

Sacamos el vector de pertenencias.

Recalculamos los centroides.

Para cada Gen{

    Para cada Cluster distinto al gen{

        Si la distancia al centroide es la minima la guardamos

    }

    Si se ha cambiado de cluster reasignar nuevo cluster

}

Mirar los patrones más cercanos a los centroides e cambiarlos por los que hay en el gen

# Descripción de cada algoritmo

## Algoritmo Genético Generacional

Se basa en generar una población intermedia, la cual sustituirá a la población actual. Como hay que decidir para cada emparejamiento si se cruza o no según una probabilidad y eso es demasiado costoso lo que hacemos es calcular el número de cruces que se hacen en total. Para las mutaciones pasa lo mismo.

## Pseudocódigo

```
Mientras (contador<10000) {
    Para número de cruces {
        Elegir 4 números aleatorios distintos entre los padres no
        elegidos.
        Realizar torneo binario entre los 4 padres
        Realizar el cruce entre los ganadores
        Sustituir los hijos por los padres
        Quitar los padres de los no seleccionados
    }
    Para número de mutaciones {
        Elegir aleatoriamente un cromosoma.
        Elegir dos posiciones aleatorias distintas
        Mutar.
    }
    Para cada cromosoma de la población intermedia {
        Si es nuevo o ha mutado {
            Evaluar cromosoma
        }
    }
    contGeneraciones++;
    Si contGeneraciones==numGeneracionesSeleccionadas{
        Calculamos el número total de búsquedas.
        Para número de búsquedas {
            Realizar búsqueda a un cromosoma aleatorio
            Evaluar
            Si mejora meterlo en la población.
        }
    }
    Si la mejor solución de la población ha desaparecido {
        Insertar mejor solución en la población intermedia
    }
}
```

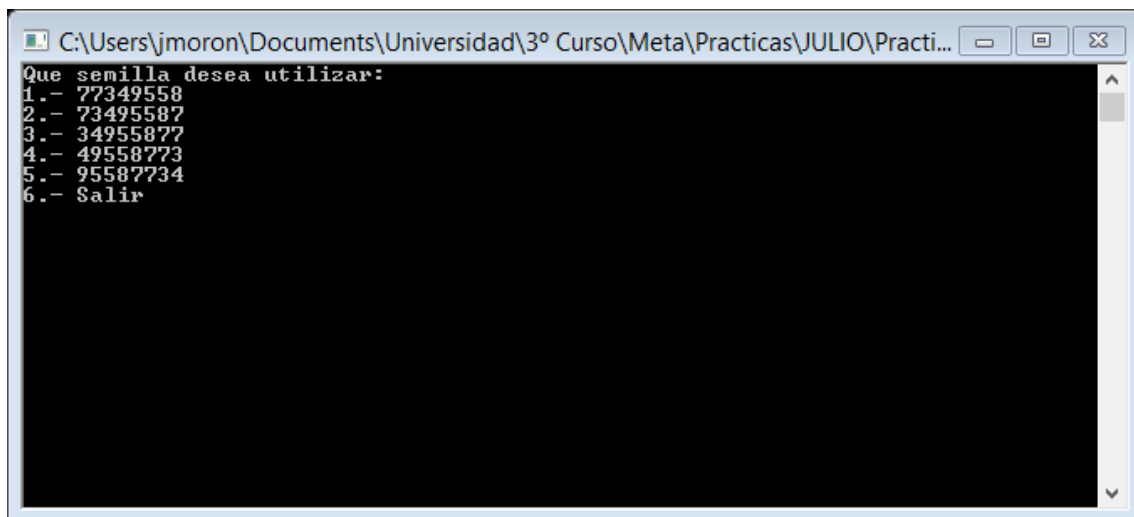
# Procedimiento para el desarrollo de la práctica

Mediante los requisitos que pone en el guión, el pseudocódigo de las transparencias de teoría y las explicaciones de los profesores, he implementado el algoritmo en C++.

## Manual de usuario

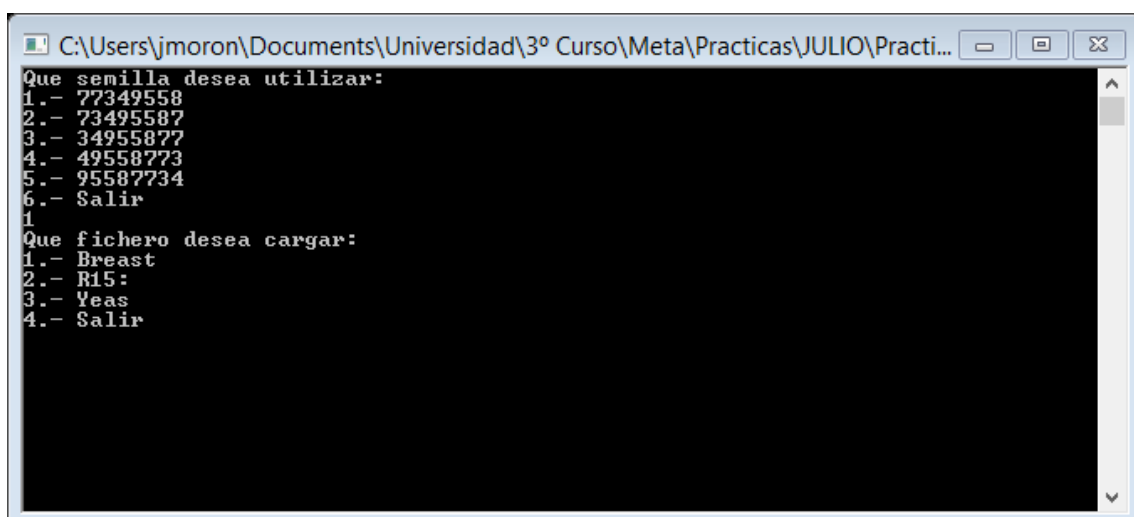
Lo primero que hay que hacer, es meter los ficheros dentro de la misma carpeta donde está el ejecutable de la aplicación.

Una vez hecho esto, el programa te pedirá que elijas una semilla:



```
C:\Users\jmoron\Documents\Universidad\3º Curso\Meta\Practicas\JULIO\Practi...
Que semilla desea utilizar:
1.- 77349558
2.- 73495587
3.- 34955877
4.- 49558773
5.- 95587734
6.- Salir
```

Una vez introducida, se pedirá que elijas el fichero de patrones que desees:



```
C:\Users\jmoron\Documents\Universidad\3º Curso\Meta\Practicas\JULIO\Practi...
Que semilla desea utilizar:
1.- 77349558
2.- 73495587
3.- 34955877
4.- 49558773
5.- 95587734
6.- Salir
1
Que fichero desea cargar:
1.- Breast
2.- R15:
3.- Yeas
4.- Salir
```

Tras elegir el fichero, se mostrará los algoritmos disponibles, elige el que quieras y el proceso se llevará a cabo:

Cuando termine, aparecerá el tiempo que ha tardado y se quedará a la espera de pulsar una tecla.

Si se pulsa volverá al principio.



# Experimentos y análisis

## Parámetros

### Patrones

Matriz de patrones que he cargado a partir de los ficheros proporcionados

### Dimensiones de cada patrón

El número de elementos que tiene cada fichero

### Número de cluster

Número de agrupaciones que haremos en el problema

### Semillas

Ejecución	Semilla
1	77349558
2	73495587
3	34955877
4	49558773
5	95587734

## Resultados obtenidos

### Kmedias

	Breast		R15		Yeast	
	<i>J</i>	Tiempo	<i>J</i>	Tiempo	<i>J</i>	Tiempo
Ejecución 1	19716,3	0,004	1368,83	0,036	45,86	0,112
Ejecución 2	19716,3	0,004	1910,01	0,022	45,8723	0,123
Ejecución 3	19716,3	0,007	1963,44	0,021	46,2371	0,209
Ejecución 4	19716,3	0,007	761,457	0,026	46,3724	0,149
Ejecución 5	19716,3	0,006	776,037	0,021	45,7844	0,134
Mejor	19716,30	---	761,46	---	45,78	---
Peor	19716,30	---	1963,44	---	46,37	---
Media	19716,30	0,006	1355,95	0,03	46,03	0,15
Desv. típica	0,00	0,002	584,36	0,01	0,26	0,04

## **Análisis de los resultados obtenidos**